UNIVERSITY OF CALIFORNIA SAN DIEGO

AN APPLICATION OF MACHINE LEARNING TO BAD PAGE PREDICTION IN MULTILEVEL FLASH

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Electrical and Computer Engineering (with a specialization in Communication Theory and Systems)

by

Navya Sree Prem

Committee in charge:

Professor Paul H. Siegel, Chair
Professor Alon Orlitsky
Professor Nuno M. Vasconcelos

2019

The thesis of Navya Sree Prem is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California San Diego

2019

DEDICATION

To my Guru, dad, mom, brother, family and friends.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I thank my advisor, Professor Paul H. Siegel, for his inspiring guidance and constant encouragement throughout my work. It has been a privilege and honor for me to have worked under his guidance. My thanks to the STAR group members for their feedback and help. I would like to thank the collaborators at Seagate for their suggestions and feedback during the project. I also want to convey my sincere gratitude to my peers at UCSD for their inputs throughout my graduate studies. Not least of all, I am greatly indebted to my family for their undying support and their unwavering trust in me.

ABSTRACT OF THE THESIS

AN APPLICATION OF MACHINE LEARNING TO BAD PAGE PREDICTION IN MULTILEVEL FLASH

by

Navya Sree Prem

Master of Science in Electrical and Computer Engineering (with a specialization in Communication Theory and Systems)

University of California San Diego, 2019

Professor Paul H. Siegel, Chair

Flash memory is prone to failures as the number of program-erase cycles increase. These physical failures in the flash result in an increase in the bit error rates. Once the bit error count exceeds a certain threshold the error correction engines are incapable of correcting the error without adversely impacting the system performance, or may even fail entirely. This leads to an interest in learning the behavior of error count increase and page failure in the flash memory and obtaining an ability to make failure predictions. We tackle this problem using a machine learning approach. However, standard machine learning techniques may not work well with the

particular data in hand. This is because the error counts are collected from actual flash memory and one can expect to see more pages with a lower error count than pages with a higher error count. This feature of the dataset leads to a formulation of our goal in terms of a classification problem with significant class imbalance in the underlying data. We have investigated various classification methods that address such class imbalance. Among those considered are cost-sensitive boosting techniques, bagging procedures, bagging ensemble support vector machines (SVMs) and cost-sensitive neural networks.

Chapter 1

Introduction

Solid State Drive (SSD) memory, more popularly known as flash memory, is a form of storage medium which has no moving parts, unlike magnetic storage devices. Due to this reason there is a reduction in the mechanical errors, but this does not ensure an error-free storage medium. The solid state memories are prone to failures due to wear and tear of the floating gates, gradual charge leakage, inter-cell interference, etc. It is a well-known fact that the floating gate cells wear out with the increase in number of program erase (PE) cycles, resulting in an increase in bit error counts in the memory. An error correction code (ECC) engine is commonly used to detect and correct the bit errors. The number of bits correctable by the ECC engine is dependent on the number of parity bits attached to the data. An ECC with a higher correction capability would result in an increase in the complexity and latency. Nonetheless, as the program erase cycles increase the error counts can exceed the correction capabilities of the ECC engine and once the number of errors increases above a certain threshold the capability to retrieve the data is lost.

Conventionally to mitigate the loss of data due to the increase in error counts in parts of the memory, a hard threshold on the number of PE cycles is used to retire a page that crosses the threshold. The threshold on the number of PE cycles is obtained using the data collected from the memory and calculating the average behavior of the error curves. A major disadvantage of

using a single threshold on all the pages in the memory stems from the fact that there are different characteristic behaviors across the drive. A page can have a faster or slower degradation curve than the average. In the case of the outlier pages with a higher degradation rate than average, the data will be lost.

This is the motivation to devise more robust methods to learn and predict errors. Machine learning is used in a wide variety of applications to predict and make decisions using characteristics learned from a dataset. A natural answer to the failure prediction problem is to use the machine learning algorithms to learn the characteristics from the error curves to predict the outliers in advance. The problem of failure prediction in flash deals with a different dataset than the conventional dataset. The underlying problem of classifying the pages as normal pages and outlier pages has to deal with an imbalanced number of samples in different classes. This project aims at investigating the different techniques catered towards classification problems with class imbalance.

The remaining sections of the report is organized as follows. Chapter 2 explains the structure of flash and defines the various data labeling performed on the dataset. Chapter 3 describes the machine learning techniques using ensemble learners such as AdaCost, Bagging and SVM ensembles. Chapter 4 describes the neural network based architecture and various cost-sensitive convolutional neural networks (CNNs). Chapter 5 explains the network architecture used in learning the time series data. Chapters 6 and 7 present the results and conclusions, respectively.

Chapter 2

Flash Memory Dataset

2.1 Flash Memory

TLC NAND flash is a solid state memory comprising NAND cells capable of storing three bits per cell. The smallest unit of read and program in the flash memory is defined as a word-line and the smallest unit of erase operation is defined as block. A block comprises certain number of word-lines and each word-line contains three pages - lower page, middle page and upper page. Prior to programming, an erase operation needs to be performed at the location. For the purpose of ECC engines the pages are further divided into frames which are the inputs to the ECC engines for correcting the errors. A page is approximately 17K bytes and is divided into 17 codewords for ECC.

The wear and tear on the memory can depend on the physical location or interference from adjacent cells. The memory errors can be attributed to various reasons like program disturb, read disturb, retention, etc. All the failures result in an increase of the bit errors in the frames. A bad frame at some PE cycle is defined as a frame with bit errors above the correction capability. A good frame at some PE cycle is defined as a frame with bit errors below the correction capability. Bad page is defined as a page containing one or more bad frames and a good page is defined as a

page with no bad frames.

2.2 Dataset

The data set used in the project is collected from a 1Xnm TLC memory. The pages are programmed with pseudo random data and erased before the next program cycle. A single cycle consists of a program and erase of the memory. After every 100 PE cycles from 4000 until 10,000 PE cycles the pages are read and the data read from the memory is compared with the programmed data. The bit errors per frame and total error per page are computed and stored. A total of 4285 word-lines are programmed, erased and read for the above mentioned PE cycles. Since each word-line consists of 3 pages and each page consists of 17 frames, the dataset contains error rates for 218535 frames for every 100 PE cycle from 4000 to 10000 PE cycle. The total dataset is approximately 119MB.

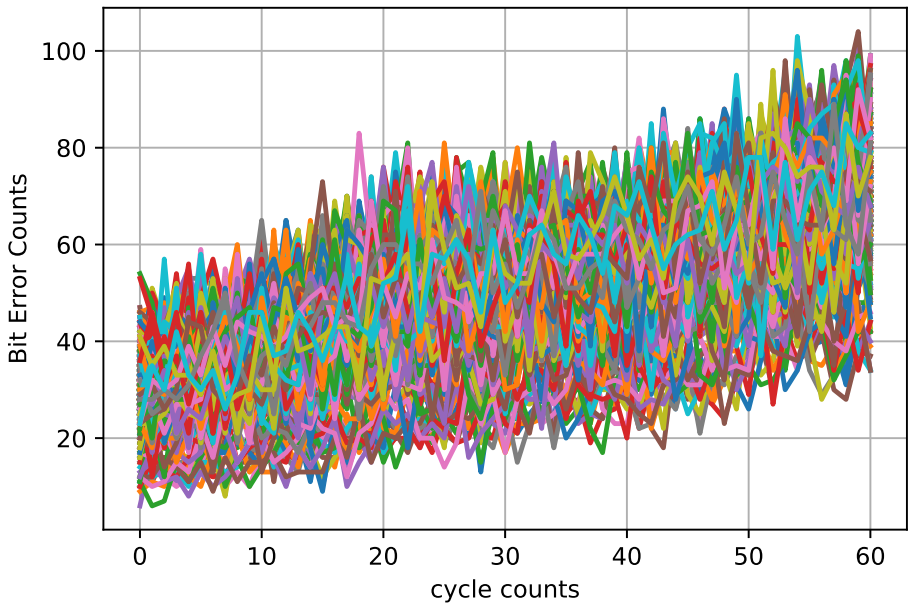

Figure 2.1: Bit error counts per frame

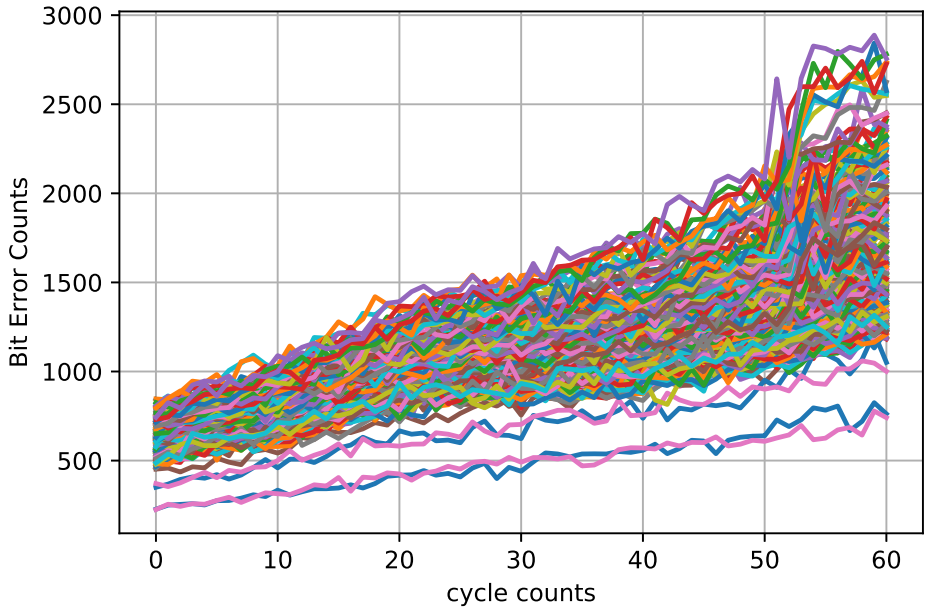

Figure 2.2: Bit error counts per page

Figure 2.1 shows the plot of error curves for different frames and Figure 2.2 plots the error curve for different pages. As previously mentioned, it can be observed that the bit error curves are not the same for all word-lines. Figure 2.3, Figure 2.4 and Figure 2.5 are the plots of the bit error counts for frames belonging to lower page, middle page and upper page for different PE cycles.

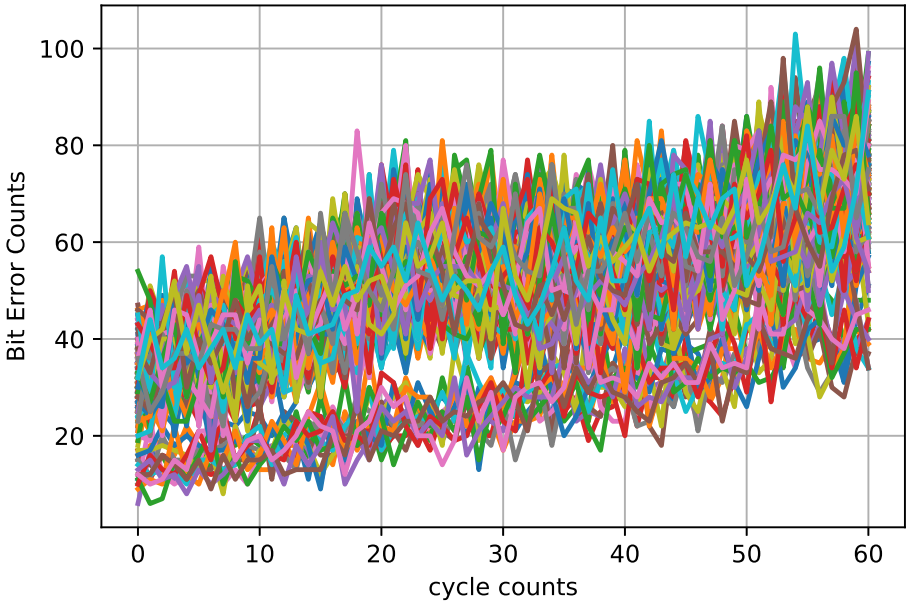

Figure 2.3: Bit error counts per frame on lower page

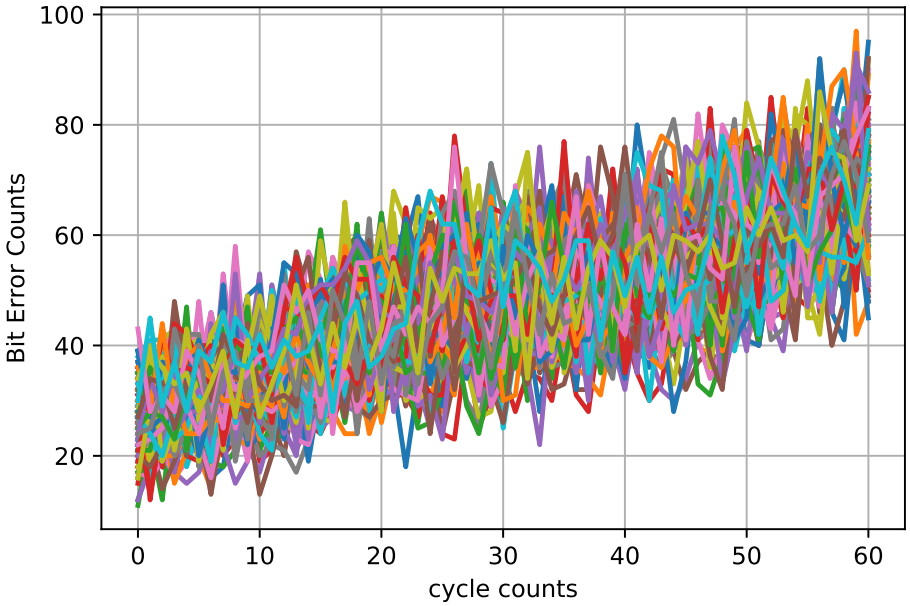

Figure 2.4: Bit error counts per frame on middle page

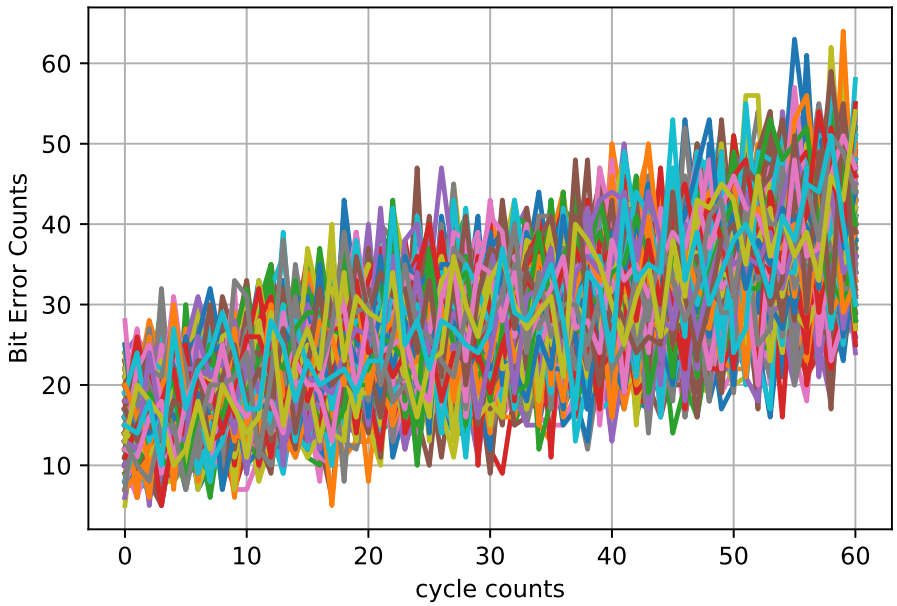

Figure 2.5: Bit error counts per frame on upper page

2.3 Data labeling

The data collected from the memory consists of bit error counts for each frame for every 100 PE cycles. To utilize the data for the classification problem, it is necessary to label the data into different classes. The two classes, as defined earlier, are good page/ bad page or good frame/bad frame. For the purpose of the work, labeling of the collected data into bad/good is performed in various ways. The different labelings serve to learn certain behaviors of the flash and are explained below.

It is important to understand the average behavior of the error counts in a frame. In many scenarios, there are stronger ECC engines present to correct frames containing more errors. These engines are normally powered down since they have a higher power or clock cycle requirements. The knowledge of average behavior can be beneficial in designing schemes to activate the higher ECC engines and avoid using the less powerful ECC engine. *Avg* bad frame is defined as a frame

with average error within a particular PE cycle windows: $[T_1 : T_1 + W_1]$ higher than a threshold, where T_1 is the start PE cycle and W_1 is the window size. The data is labeled as bad and good frames using the above mentioned *Avg* criterion.

Next, there is a need to understand the sporadic bit errors that can occur in frames. These error rates are higher than the correction capability of the stronger engines present in the memory. Understanding such error behaviors are a necessity since if these frames are not predicted correctly, the programmed data can be lost. *Any* bad frame is defined as a frame with bit error counts higher than a threshold in a particular $[T_1 : T_1 + W_1]$ windows of PE cycles. The data is labeled as bad or good frames using the above mentioned *Any* criterion.

The third labeling method uses *Avg* or *Any* to label a page as good or bad. The frames are labeled for different PE cycle windows, for example $y_{T_1}, y_{T_2}, y_{T_3}$ are the labels for the frames during the window $[T_1 : T_1 + W_1]$,$[T_2 : T_2 + W_1]$ and $[T_3 : T_3 + W_1]$, respectively.The classification problem is similar to classifying the frames as good/bad frames for the particular PE cycle window. This aims at understanding the correlation between the error curves and different PE cycle windows.

The input variables for all the classifiers are the bit error counts from a window $[T_0 : T_0 + W_0]$ taken at every 100 PE cycles where $(T_0 + W_0) < T_1$. The inequality ensures the classifier predicts without the knowledge of bit error counts used for the labeling. For the third labeling method, the inputs to the classifiers are error counts from a window $[T_i : T_i + W_0]$ and the variable T_i denoting the starting PE cycle of the data for a certain number of windows.

2.4 Characteristics of Dataset

The flash drives are physically designed to reduce the number of bit errors in the cell from the various disturbs. This implies the number of pages or frames with an abnormal number of errors that are beyond the correction capabilities is small. As expected, after labeling the dataset

obtained from the physical memory the number of instances in each class is not balanced. The number of data points belonging to the class of good frames significantly larger than the number of data points belonging to the class of bad frame. In the context of this classification problem, it is more important to learn and classify the bad frames which form the minority class in this dataset. A classification problem where the number of samples in a particular class is significantly higher(majority class) than the samples from the other class(minority class) is broadly defined as a class imbalance problem, with the class of interest being the minority class. Conventionally with a huge imbalance, the classifier leans towards learning the class with a higher number of instances rather than the class with a lower number of instances. Such classifiers can achieve high accuracy by correctly classifying the majority class data but not a high accuracy when considering the classification of the minority class. Thus, it is important to re-define the metric of the classifiers when working on an imbalanced dataset. Instead of overall accuracy, a trade-off between accuracy and recall is chosen as metric for the imbalanced class problem. In the project, *Accuracy* is defined as the percentage of total number of correctly classified points and *Recall* is defined as the percentage of bad pages correctly classified out of the total number of bad pages.

Chapter 3

Ensemble Learners

As mentioned in the previous chapter, the dataset has imbalance in classes as an attribute. There are various methods found in the literature for handling imbalanced dataset. From an initial analysis of the error curves, a few observations can be made. First, the bit errors have a general increasing trend and, second, a frame with a higher rate of increase in the error has a higher probability to cross the threshold. The bit error counts for a frame have a high correlation with each other, and thus the first set of algorithms that are studied belong to the class of ensemble weak learners. The learning techniques using ensemble learners can be broadly divided into three groups: algorithmic approaches, data distribution approaches and a hybrid of these approach. The techniques that are implemented to learn the bad page classifiers are: cost sensitive boosting, bagging, SVM ensemble. The different techniques are explained in the sections below.

3.1 AdaBoost

AdaBoost[2], short for adaptive boosting, is a machine learning algorithm which uses weak learners in order to learn the classification problem. AdaBoost uses the complete dataset to train the weak classifiers serially with adaptively tweaking the weights assigned to each sample in the dataset. Each sample in the training set is assigned a weight. Initially, equal weights are

assigned to all the samples representing equal importance to all the samples. After each iteration of learning a weak classifier, the weights are increased for the misclassified instances and the weights associated with the correctly classified instances are reduced. The objective is to give more importance to learn the misclassified points in the subsequent iteration. Further, another factor(α_t) is assigned to each classifier which depends on the overall accuracy of the classifier. This factor along with the classifiers are used to classify a new sample in the test phase. The weight update for a sample x_i with the correct label y_i at iteration t when the weak learner is h_t is given by,

$$W^{t+1}(i) = W^t \frac{\exp(-\alpha_t h_t(x_i) y_i)}{Z_t} \tag{3.1}$$

$$\alpha_t = \frac{1}{2} \log \frac{\sum_{i,h_t(x_i)=y_i} W^t(i)}{\sum_{i,h_t(x_i)\neq_i} W^t(i)} \tag{3.2}$$

where Z_t is the normalization factor for the weights and α_t is the factor reflects the accuracy of the classifier h_t. A detailed flowchart is shown in Figure 3.1.

AdaBoost algorithm considers all the instances equally without discrimination made with respect to the class. AdaBoost is designed to achieve better accuracy, and with an imbalanced dataset, the algorithm can be biased to better learn the majority class that the minority class. Thus, for the class imbalance problem modifications to the algorithm are made to incorporate a cost per class into the computation. The algorithms which incorporate such cost functions depending on the class are implemented and tested for the bad frame detection dataset. The algorithms from literature that are implemented, AdaC1[3], AdaC2[3] and AdaCost[4], are explained in detail below.

3.1.1 AdaC1- Exponential

AdaC1[3] is a modification to AdaBoost algorithm. As mentioned in the previous section, AdaBoost uses a weight updating function to learn the misclassified points in each iteration

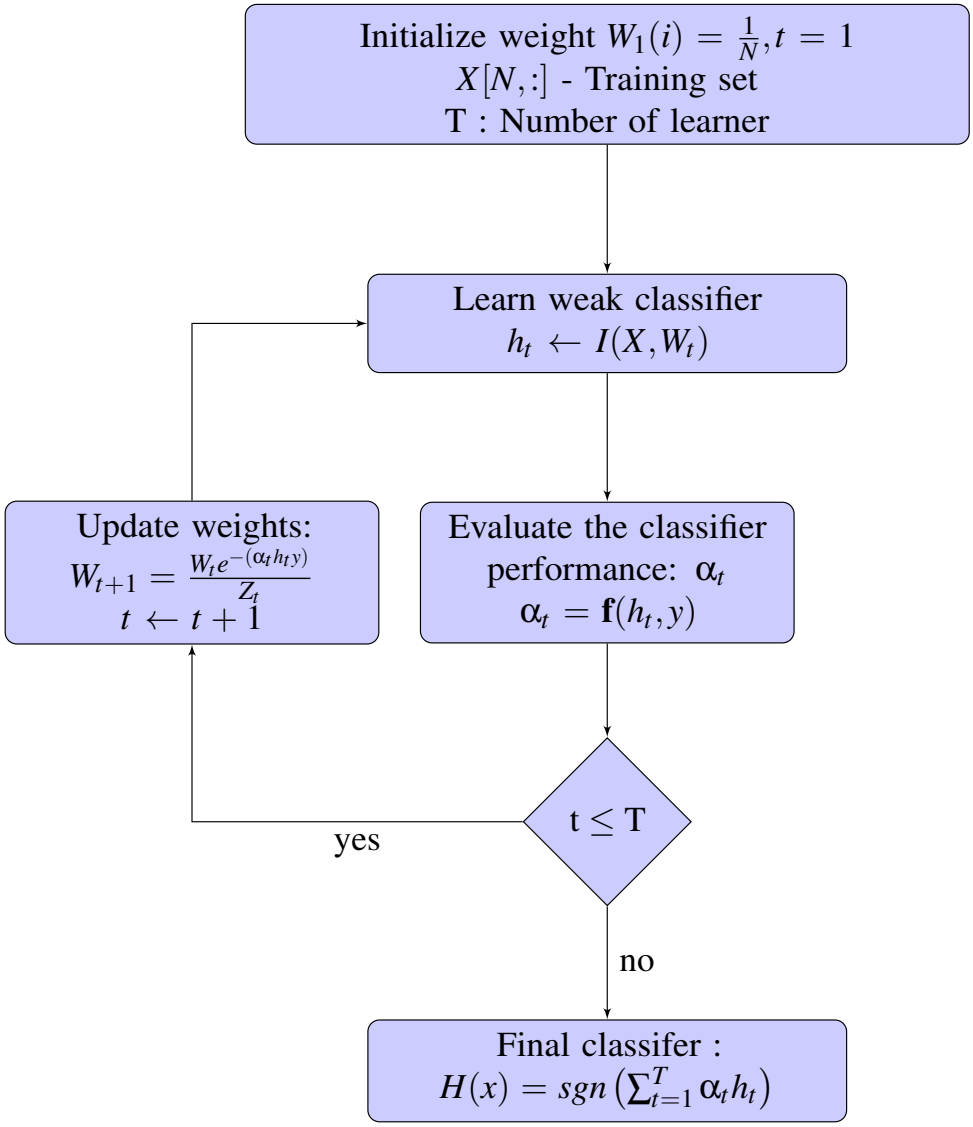

Figure 3.1: AdaBoost - Algorithm flowchart

without distinguishing the samples from different classes. In this paper, a cost is assigned to each sample depending on the class. The cost is used to differentiate the importance of correctly classifying samples belonging to different classes.

Consider a $2-$class classification problem where class-0 is good page and class-1 is bad page, and define $C_i \in [0,\infty)$ where C_0, C_1 represent the cost of the respective classes. AdaC1 introduced the cost per class in the weight updating function of AdaBoost as an exponent. The weight updating function is:

$$W^{t+1}(i) = \frac{W^t(i)\exp(-\alpha_t C_i h_t(x_i) y_i)}{Z_t} \tag{3.3}$$

$$Z_t = \sum_i W^t(i)\exp(-\alpha_t C_i h_t(x_i) y_i) \tag{3.4}$$

where $W^t(i)$ is the weight associated with the i^{th} sample during the t^{th} iteration, $h_t(x_i)$ is decision of x_i sample by the weak learner at iteration t, y_i is the correct label and α_t is computed to minimize Z_t. The inequality holds from [5]

$$\sum_i W^t(i)\exp(-\alpha C_i y_i h(x_i)) \leq \sum_i W^t(i)\left(\frac{1+C_i y_i h_t(x_i)}{2}e^{-\alpha} + \frac{1-C_i y_i h_t(x_i)}{2}e^{\alpha}\right) \tag{3.5}$$

α_t is obtained by zeroing the first derivative of 3.5 with constraint $\alpha_t > 0$,

$$\alpha_t = \frac{1}{2}\log\frac{1+\sum_{i,y_i=h_t(x_i)} C_i W^t(i) - \sum_{i,y_i\neq h_t(x_i)} C_i W^t(i)}{1-\sum_{i,y_i=h_t(x_i)} C_i W^t(i) + \sum_{i,y_i\neq h_t(x_i)} C_i W^t(i)} \tag{3.6}$$

3.1.2 AdaC2- Linear

AdaC2[3] similar to AdaC1, integrates the cost for different classes into the weight update function. In AdaC2 the cost is introduced as a linear factor outside the exponent. The weight

update equation is:

$$W^{t+1}(i) = \frac{C_i W^t(i) exp(-\alpha_t C_i h_t(x_i) y_i)}{Z_t} \tag{3.7}$$

$$Z_t = \sum_i C_i W^t(i) exp(-\alpha_t C_i h_t(x_i) y_i) \tag{3.8}$$

α_t for AdaC2 is computed similar to AdaC1 to minimize Z_t. Using the same inequality with the new weight updating equation, α_t that minimizes Z_t can be computed as

$$\alpha_t = \frac{1}{2} \log \frac{\sum_{i, y_i = h_t(x_i)} C_i W^t(i)}{\sum_{i, y_i \neq h_t(x_i)} C_i W^t(i)} \tag{3.9}$$

Since $\alpha_t > 0$, 3.9 implies,

$$\sum_{i, y_i = h_t(x_i)} C_i W^t(i) > \sum_{i, y_i \neq h_t(x_i)} C_i W^t(i) \tag{3.10}$$

From eq[3.10], the weak classifier h_t learnt is better than a random classifier.

3.1.3 AdaCost

AdaCost[4] introduces the cost per class as a cost adjustment function φ. The cost adjustment function increases the weight more when the instance is misclassified and decreases the weight less if the instance is classified correctly. Consider C_i is the cost of misclassifying a sample, the paper introduces $\varphi_+ = -0.5C_i + 0.5$ and $\varphi_- = 0.5C_i + 0.5$ as the cost adjustment function. Using the cost adjustment function, the weight updating function is modified as,

$$W^{t+1}(i) = W^t(i) \exp(-\alpha_t h_t(x_i) y_i \varphi_{sign(h_t(x_i) y_i)}) \tag{3.11}$$

The value of α_t is calculated as,

$$\alpha_t = \frac{1}{2}\log\frac{1+\sum_i W^t(i)\exp(-\alpha_t h_t(x_i)y_i\varphi_{sign(h_t(x_i)y_i)})}{1-\sum_i W^t(i)\exp(-\alpha_t h_t(x_i)y_i\varphi_{sign(h_t(x_i)y_i)})} \tag{3.12}$$

Note, the cost-sensitive algorithms AdaC1 and AdaC2 reduce to AdaBoost when the weights are set to be equal but AdaCost does not reduce to AdaBoost.

3.2 Bagging

Another approach to deal with class imbalance is using bagging[6] ensembles. The essence of bagging ensembles is obtaining a new dataset from the original dataset using resampling procedures defined with the objective to create a balanced dataset. There are various algorithms in the literature dealing with the class imbalance problem with data pre-processing. Bagging technique is comparatively simpler than algorithmic classifiers like boosting since there are no weight updates to be computed. Different types of bagging can be broadly classified into two types, OverBagging and UnderBagging. For implementation purposes, ensemble decision trees are implemented for bagging. The next sections explain UnderBagging and OverBagging.

3.2.1 UnderBagging

UnderBagging[7] uses the idea of preprocessing the dataset to mimic a balanced dataset. Instead of performing the classification on the actual imbalanced dataset, the classifier is trained on a sub-sampled training data with respect to the majority class. Since the number of samples are greater in one class (majority class) than the number of samples in the other class (minority class), the under-sampling is applied to the majority class. This creates a more balanced data distribution for the training procedure. A majority vote is taken to make the prediction during the test phase. One should not ignore the fact that while subsampling there is a probability of losing

some useful majority class samples.

UnderBagging idea is used in various papers with some variation, for example, asymmetric bagging [8], roughly-balanced bagging[9], etc. Roughly-balanced bagging uses a negative binomial distribution in each iteration to determine the number of majority class samples drawn.

3.2.2 OverBagging

In contrast to UnderBagging, OverBagging[7] uses oversampling method to obtain a balanced data distribution. Thus, instead of sampling randomly from the dataset, an oversampling is performed on the minority class before training classifiers. Note that OverBagging procedures result in an increase in the total number of samples since all the majority class instances are included in training along with replication of the minority class instances. Oversampling of the minority class is performed using SMOTE algorithm which is *synthetic minority oversampling technique*. SMOTE is a method to create new minority class instances by interpolating from minority class instances that lie close together. Certain OverBagging algorithms use both resampling and SMOTE to create minority class samples. A fixed number of minority class instances are randomly sampled with replacement from the dataset and the rest minority class instances are generated by SMOTE algorithm. One disadvantage with OverBagging procedure is the increase in complexity with respect to the increased number of samples during the training process.

3.3 SVM Ensembles

The paper[10] proposes an ensemble learning technique using support vector machine(SVM) and data processing. SVM[11] was proposed to obtain a decision surface that separates the samples into the classes. The decision function is given by

$$y = sign\left(\sum_{i=1}^{N} y_i \alpha_i K(x, x_i) + b\right) \tag{3.13}$$

where x is a d-dimensional test sample, N is the number of training points, x_i is the i^{th} training sample and $K(x,x_i)$ is the kernel function. The SVM is defined with parameters $\alpha = \{\alpha_1, ..., \alpha_N\}$ and b. The values of α_i are obtained by solving a quadratic programming problem,

$$\min_{\alpha} \left(-\sum_{i=1}^{N} \alpha_i + \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \right) \tag{3.14}$$

$$\text{subject to} \sum_{i=1}^{N} \alpha_i y_i = 0 \qquad \text{and} \quad 0 \leq \alpha_i \leq C \quad \forall i \tag{3.15}$$

Instead of removing data points or adding duplicate data points as performed in the bagging procedure, SVM ensembles tackle the class imbalance problem without changing the data distribution. The strategy partitions the majority class samples into a certain number of sets. The number of sets define the number of SVMs that are trained. Consider the majority class is partitioned into K different sets and each set is combined with minority class to form a training set. Then K SVMs are trained independently on the K training sets. The complete minority class data points are used in training all the K different SVMs. Figure 3.2 represents the architecture.

The final classification is performed using a combination strategy. Either the class is determined using a majority vote from the K classifiers outputs or the class is determined by evaluating the binary classifier on the sum of the scores(real valued outputs) from individual SVMs. The final decision can be written as a function of outputs from K SVMs, $f_{output}(x) = F(f_1(x), f_2(x), ...f_k(x))$.

There are various advantages of using SVM ensembles. Unlike UnderBagging SVM, there is no loss of information in SVM ensembles and unlike OverBagging SVM, the complexity is maintained. SVM ensembles have a better performance with respect to minority class compared to individual SVM because as previously mentioned, a single SVM is skewed towards learning the majority class data and ignoring the characteristics of the minority class data.

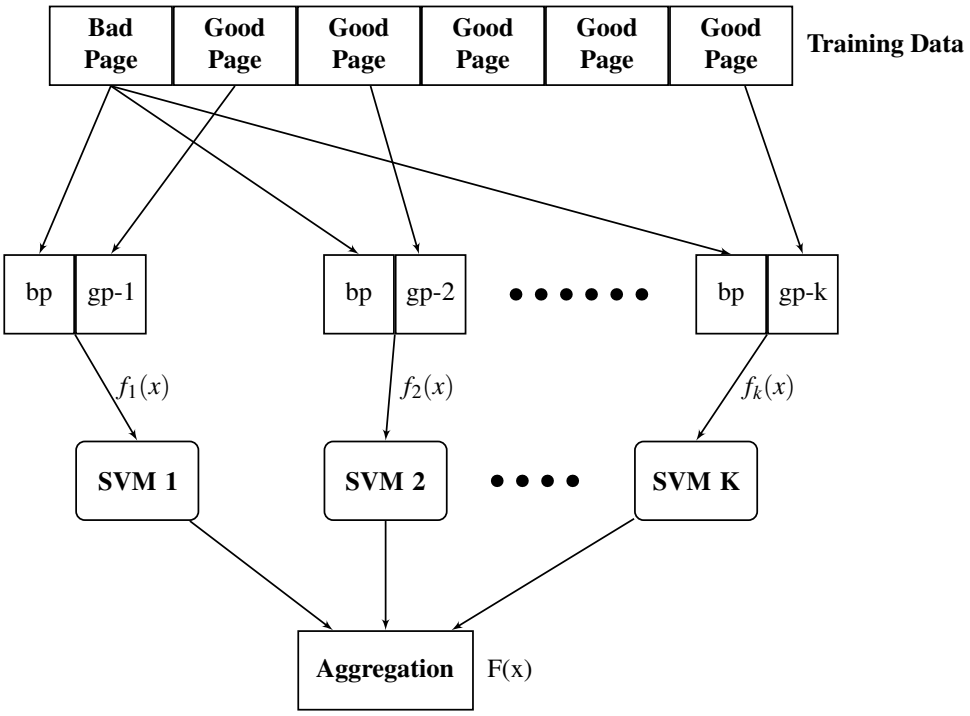

Figure 3.2: SVM ensembles

Chapter 4

Neural Network

4.1 Convolutional Neural Network

After understanding the classical machine learning techniques using ensemble learners and evaluating the performance of the models, a natural question that arises is whether neural network based machine learning techniques can learn the imbalanced data set without introducing a cost function. This provides a perspective on the achievable performance with increasing complexity. A CNN with three layers is implemented to understand the performance of a neural network on the dataset. We consider the dataset as a one dimensional time series of bit error counts. The CNN includes three one dimensional filters that operate as feature extraction units with a rectifier unit attached to each and a pooling is performed on the last layer. The final layers of the CNN are two fully connected layers with the last fully connected layer performing the binary classification. A standard gradient descent is used as the optimizer with a cross entropy loss. The implemented CNN architecture is shown in the Figure 4.1 below.

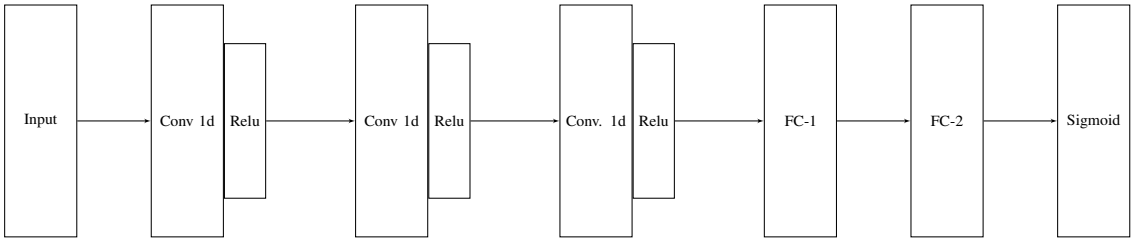

Figure 4.1: CNN architecture

4.2 Bagging - CNN

Prior to introducing a cost factor in the training process, data processing techniques are explored with the network mentioned in the previous section. Similar to the bagging procedures mentioned previously, oversampling and undersampling are performed on the minority and majority class, respectively.

Oversampling addresses the class imbalance problem by resampling the minority class samples in order to create a training dataset comprising an equal number of samples in every class. Oversampling the minority class can be performed either by duplicating a few of the data samples or using a more advanced algorithm like SMOTE[12] to create synthetic samples. As previously noted, oversampling increases the number of data samples leading to an increased complexity in the training. Oversampling can result in network overfitting with respect to the minority class.

Undersampling, similar to oversampling, changes the data distribution by subsampling the majority class samples to obtain a training dataset with an equal number of samples from each class. To eliminate the loss of important samples holding crucial information about the class during subsampling, [13] introduces a routine to locate the redundant samples. Redundant samples are defined as the samples whose part can be taken over by other samples. Even with a careful subsampling of the majority class the performance of the network depends on how efficiently the algorithm discards the insignificant samples.

4.3 Output Cost CNN - Threshold Moving

Classifying the minority class samples accurately can be achieved with shifting the decision threshold towards the particular class. The method explained in this section implements a weighted output decision function during the test phase with no modification for the training. $C_{[i,k]}$ is introduced as the cost of classifying a sample i to the class k. The cost of correct classification is set to one and cost of misclassifying a sample is determined according to the importance of the class. The cost factor defined above is used during the classification of a test sample.

Let O_i denote the real-valued output from the neural network performing K-class classification with $\sum_{i=1}^{K} O_i = 1$, and $0 \leq O_i \leq 1$. The classification is defined by,

$$\hat{y}_i = \arg\max_i O_i \tag{4.1}$$

Whereas, in the threshold moving algorithm, the cost $C_{[i,k]}$ is used to modify the outputs prior to making the decision. The output O_i is updated to O_i^* as,

$$O_i^* = \eta \sum_{k=1}^{K} O_i C_{[i,k]} \tag{4.2}$$

where η is the normalization factor. The final decision is defined by,

$$\hat{y}_i = \arg\max_i O_i^* \tag{4.3}$$

4.4 Cost-Sensitive CNN

The final approach explored in this project, using CNN, modifies training into a cost-sensitive training procedure by including a cost per class in computing the training loss. This

results in achieving a network sensitive to extracting features from the minority class. There are various methods to define the cost used in evaluating the training loss such as a fixed cost dependent on the imbalance ratio, an adaptively changing cost[14] depending on the overall training performance or an adaptively changing cost depending on the batch performance[15].

The network previously designed is trained with various fixed costs. The cost for the minority class is varied from [1.0] to [10.0] with maintaining the cost for the majority class as [1.0]. The network is trained using a cost dependent entropy loss and a standard gradient descent as the optimizer. The cost dependent entropy loss function is given by

$$Loss(y,d) = -\sum_{n} C_n \left(d_n \log(y_n) + (1-d_n)\log(1-y_n)\right) \tag{4.4}$$

where d_n and y_n are the desired label and predicted output for the sample n, respectively, and C_n is the cost associated with the class of sample n. In[15], an adaptive cost depending on the imbalance ratio and the performance of the mini-batch is introduced. The authors define a cost for each sample depending on the class and introduce the formula to calculate the cost for every iteration. The formula introduced in the paper seems to reverse the purpose of cost per class while training an imbalanced dataset. Therefore, we modified the definition to reflect the purpose of introducing cost for the majority and the minority class. C_n is defined as the cost for a sample n as in (4.5).

$$C_n = \begin{cases} \text{IR} * \exp\left(-\dfrac{G_{mean}^{batch}}{2}\right) * \exp\left(-\dfrac{Acc^{batch}}{2}\right) & \text{sample } n \in \text{minority class} \\ 1 & \text{sample } n \in \text{majority class} \end{cases} \tag{4.5}$$

IR is defined as the imbalance ratio of good frames to bad frames, G_{mean}^{batch} is the geometric mean of correctly classified minority samples (bad frames) and incorrectly classified minority samples (bad frames) and Acc^{batch} is defined as the accuracy of the current batch. G_{mean}^{batch} and Acc^{batch} are

computed as mentioned below,

$$G_{mean}^{batch} = \sqrt{\frac{TB}{TB+FG} * \frac{FG}{TG+FB}} \tag{4.6}$$

$$Acc^{batch} = \frac{TB+TG}{TB+FB+TG+FG} \tag{4.7}$$

where TB is the number of correctly classified bad frames in the batch, TG is the correctly classified good frames in the batch, FB and FG are the number of incorrectly classified good and bad frames in the batch.

Chapter 5

5.1 Time Series Data - LSTM

Long short-term memory[16] is a variant of recurrent neural network(RNN) used for classifying and making prediction on a time series data without the problem of vanishing gradient descent typically observed in training RNNs. Recurrent networks differ from a feedforward network in the fact that these networks contain a feedback loop connecting the previous decision which is often referred to as memory. Thus, the prediction at any time t is dependent on the input variables at time t and the information learned from the previous time that is preserved in the memory referred to as hidden state. A recurrent network can be mathematically expressed as,

$$h_t = f(h_{t-1}, x_t) \tag{5.1}$$

$$y_t = W h_t \tag{5.2}$$

The hidden state of the network at time t, denoted by h_t, is a function of the current input x_t and the previous hidden state h_{t-1} and the output of the network y_t is calculated using the current hidden state. Similarly, an LSTM cell can be mathematically represented using the equations

below.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{5.3}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{5.4}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{5.5}$$

$$g_t = \tanh(W_g x_t + U_g h_{t-1} + b_g) \tag{5.6}$$

$$c_t = f_t * c_{t-1} + i_t * g_t \tag{5.7}$$

$$h_t = o_t * \tanh(c_t) \tag{5.8}$$

where h_t is the hidden state at time t, x_t is the input at time t, c_t is the cell state at time t, h_{t-1} is the hidden state at time $t-1$, i_t is the input gate, f_t forget gate, g_t cell gate and o_t is the output gate. σ is the sigmoid function and $*$ denotes the Hadamard product. Figure 5.1 is a visual representation of an LSTM cell.

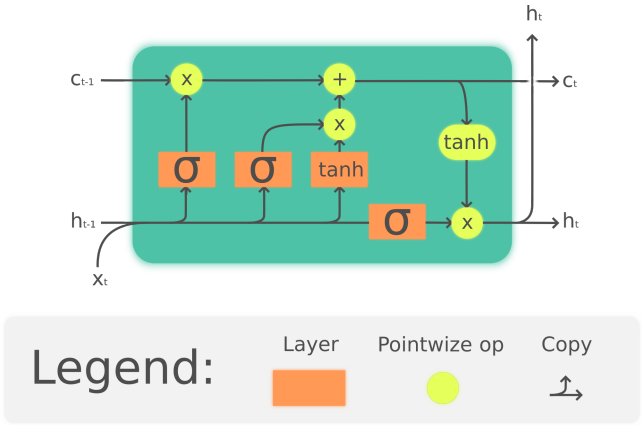

Figure 5.1: LSTM cell architecture[1]

The third labeling method mentioned in section 2.3 contains samples from adjacent PE cycle windows which can be represented as a time series data for LSTM. The bad page prediction for a later PE cycle window can utilize the information obtained from the previous PE cycle windows using a recurrent network architecture. The network architecture for bad page prediction

using the time series data consists of a single layer LSTM with 20 hidden features and two fully connected layers with the final fully connected layer outputting a single value. The mean squared loss and standard gradient descent optimizer is used while training.

Chapter 6

Results

6.1 Labeling - Avg

In this section, we will present the results on the average labeling technique using the methods explained in the previous sections. The sum of bit error counts from $[9K : 10K]$ PE cycle for every 100 PE cycle is used to label a particular frame as a good/bad frame with the threshold set as 2000 bit errors and the input variables to the classifier are the bit error counts from PE cycle $[7K : 8K]$. After labeling the 218435 frames, we obtain 195378 good frames and 44561 bad frames which is divided into 198435 training samples consisting of 176931 good and 21504 bad frames and 20000 testing samples consisting of 18447 good frames and 1553 bad frames.

We first present the results for ensemble learners using cost-sensitive AdaBoost algorithms. We vary the model parameters cost per class, number of estimators, and the maximum depth of decision trees to understand the impact of these parameters on learning the imbalanced dataset. While assigning the cost C_0 for good frames as 1, the cost C_1 for bad frames is assigned as either $1.5, 2$ or 2.5 and the maximum depth of the decision tree is set to $2, 4, 6, 8$ and 10. Figures 6.1, 6.2 and 6.3 plot the training accuracy, testing accuracy and testing recall for fixed cost and varying depth of decision trees for AdaC1 algorithm with the number of estimators as 50.

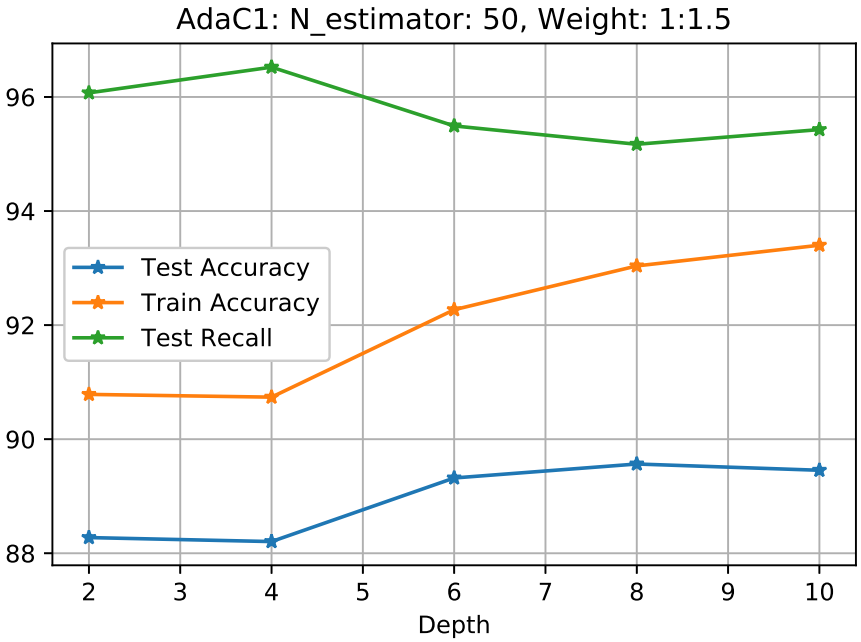

Figure 6.1: AdaC1 - Number of weak learners: 50

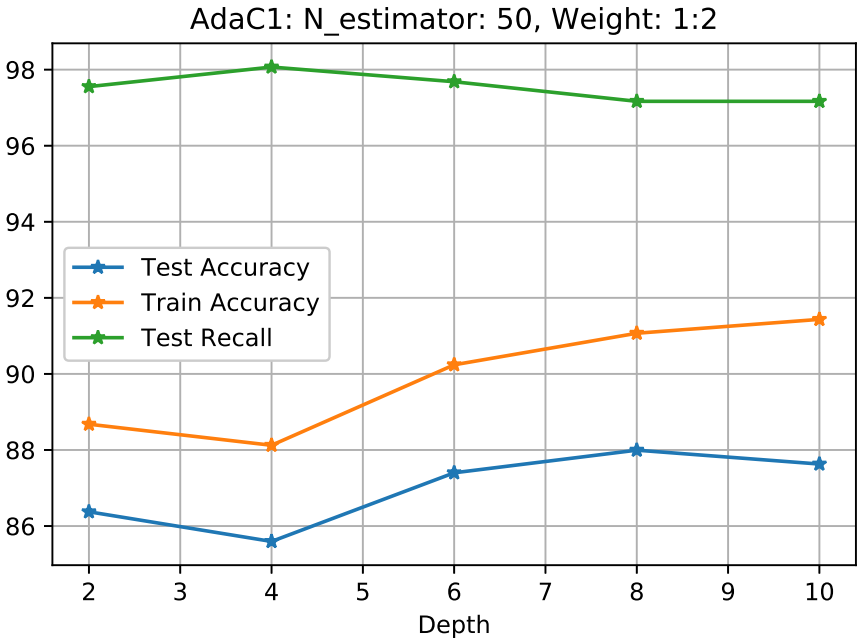

Figure 6.2: AdaC1 - Number of weak learners: 50

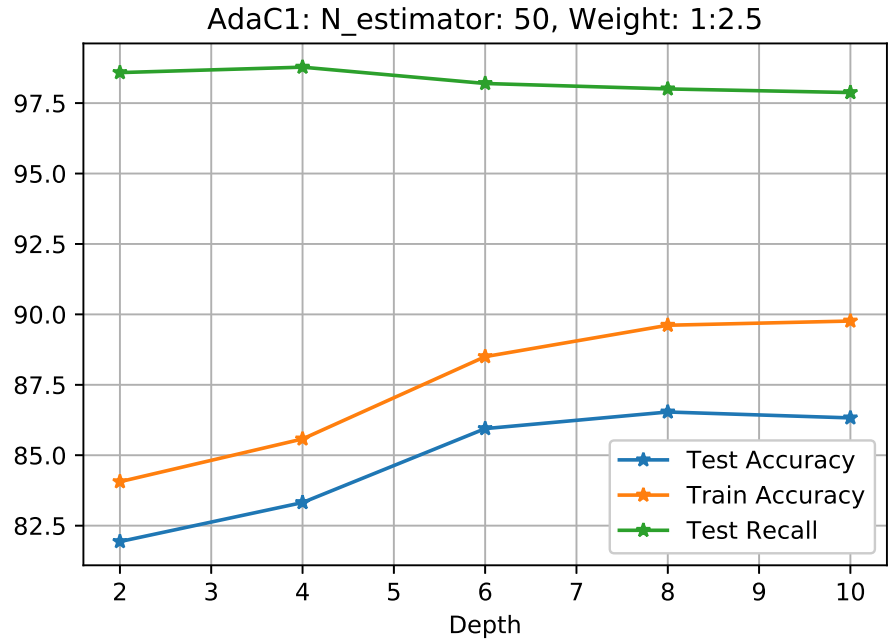

Figure 6.3: AdaC1 - Number of weak learners: 50

To understand the performance of classifiers with varying number of weak learners and cost, the number of estimators are set to $10, 25, 50, 75$ and 100 and the cost for bad frame is set to either $1.5, 2$ or 2.5 during the training process. Figures 6.4, 6.5 and 6.6 plot the training accuracy, testing accuracy and testing recall for varying cost and varying number of estimators for AdaC1 algorithm with the fixed maximum depth of the decision tree as 6.

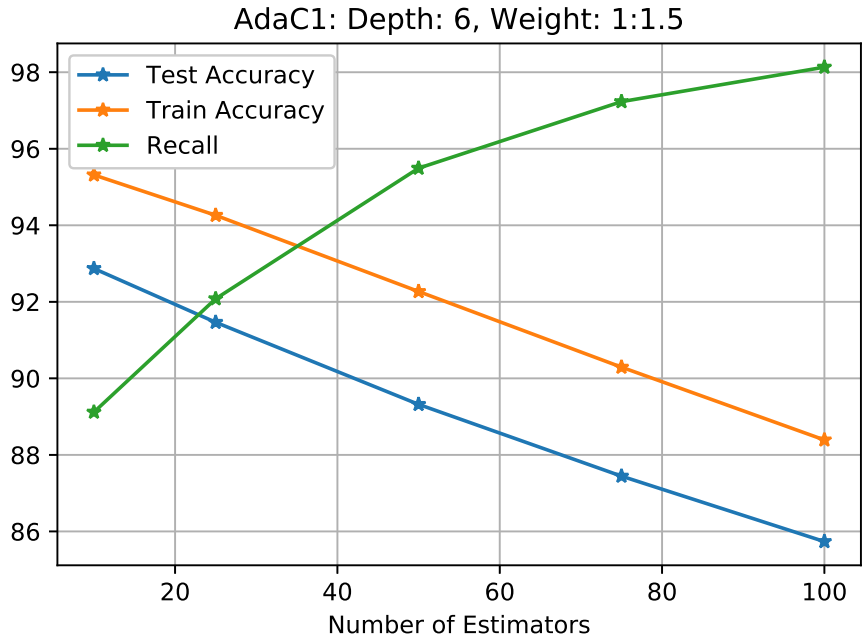

Figure 6.4: AdaC1 - Maximum depth of tree: 6

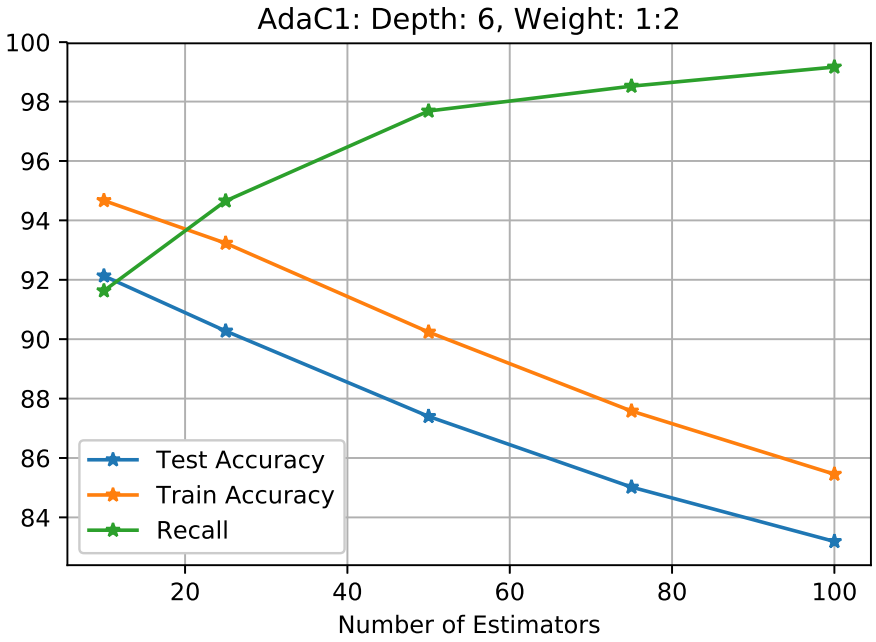

Figure 6.5: AdaC1 - Maximum depth of tree: 6

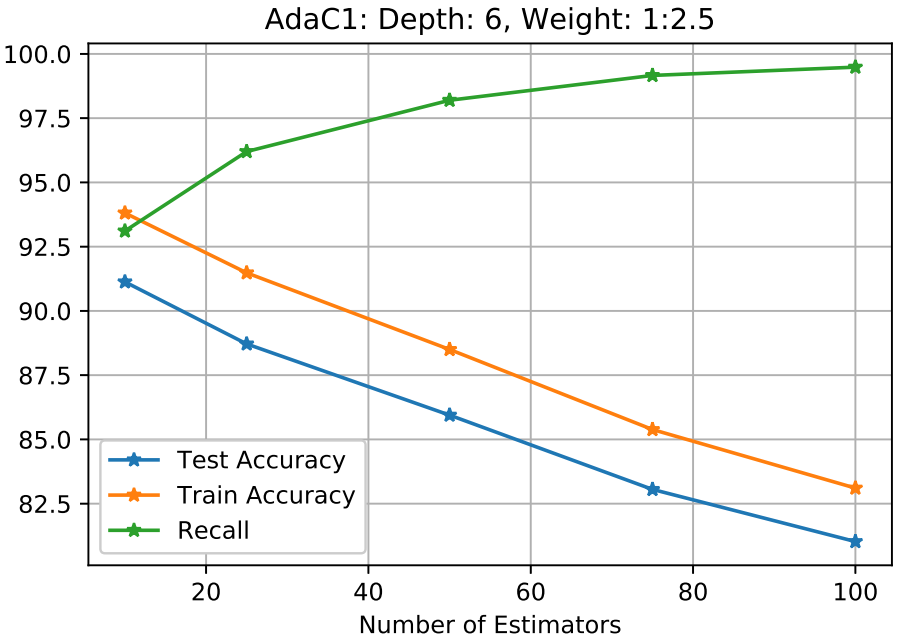

Figure 6.6: AdaC1 - Maximum depth of tree: 6

From the previous plots we make the following observations:

1) As we increase the cost for misclassifying a bad frame, the recall percentage increases.

2) As we increase the complexity of the weak learners shown by the increase in maximum depth of the decision trees, the overall accuracy of the classifier increases.

3) As we increase the number of estimators, there is a steep decrease in accuracy and steep increase in recall.

We obtain results for AdaC2 classifiers with the different values of cost, number of estimators and maximum depth as performed for AdaC1. Algorithm AdaC2 incorporates the cost factor linearly into the weight updating rule, using the cost ratios as performed for AdaC1 algorithm will result in a classifier highly skewed towards the minority class. Thus, the cost C_0 for the good frames is set equal to 1 and the cost C_1 for the bad frames is set to either $1.1, 1.3$ or 1.5 and the maximum depth of the decision tree is set to $2, 4, 6, 8$, and 10. Figures 6.7, 6.8 and

6.9 plots the training accuracy, testing accuracy and testing recall for the mentioned parameter variations and the number of estimators set to 25.

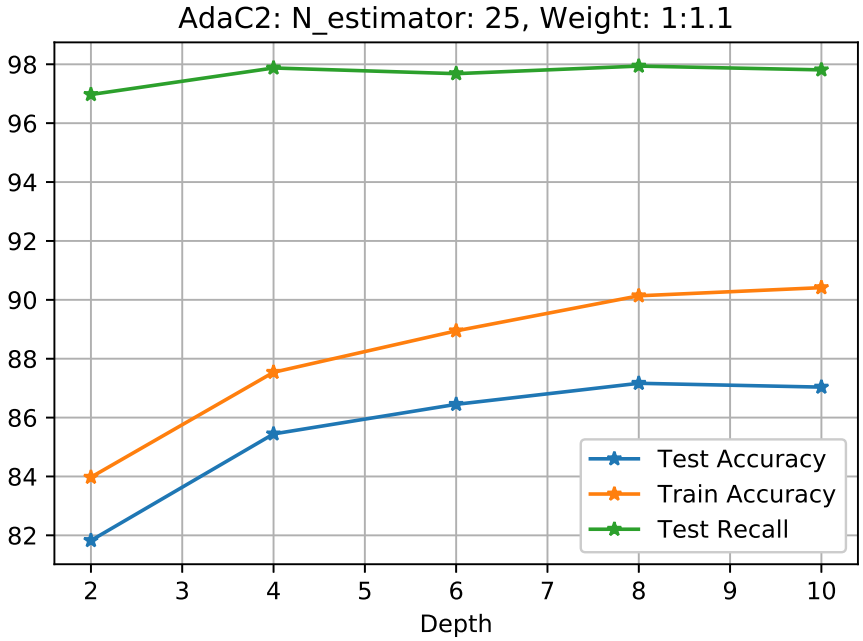

Figure 6.7: AdaC2 - Number of weak learners: 25

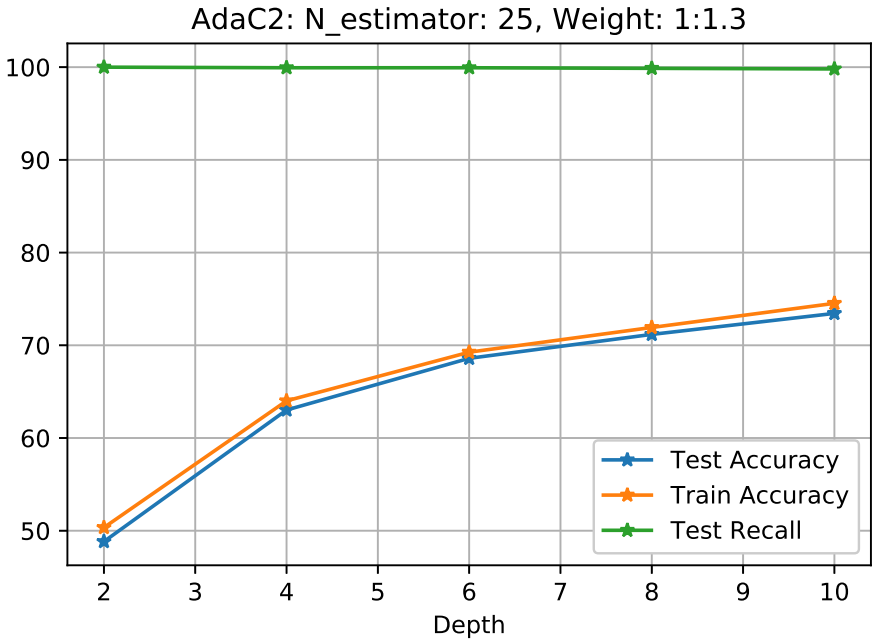

Figure 6.8: AdaC2 - Number of weak learners: 25

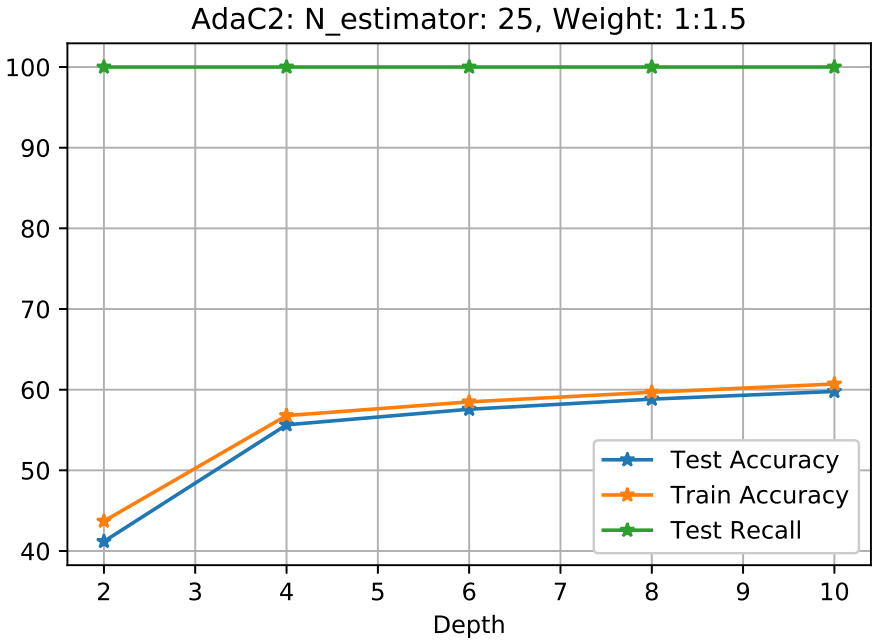

Figure 6.9: AdaC2 - Number of weak learners: 25

Figures 6.10, 6.11 and 6.12 plot the classifier performance for varying number of weak learners $10, 15, 25, 35$ and 50. The cost for bad frames set to either $1.1, 1.3$ or 1.5 with the maximum depth of decision trees fixed to 4.

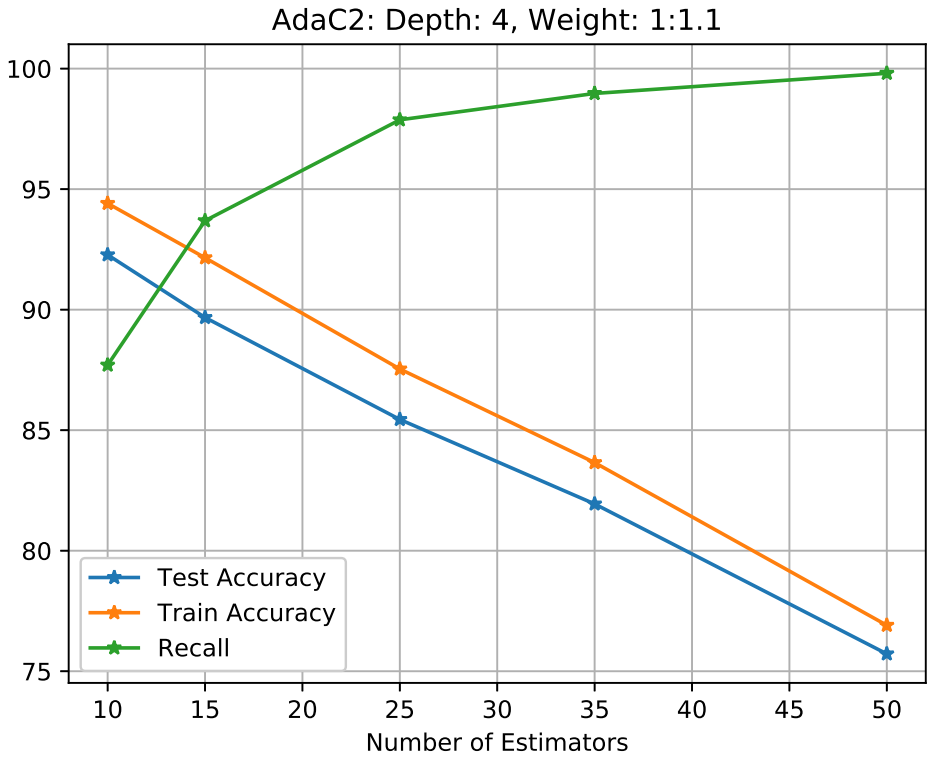

Figure 6.10: AdaC2 - Maximum depth of tree: 4

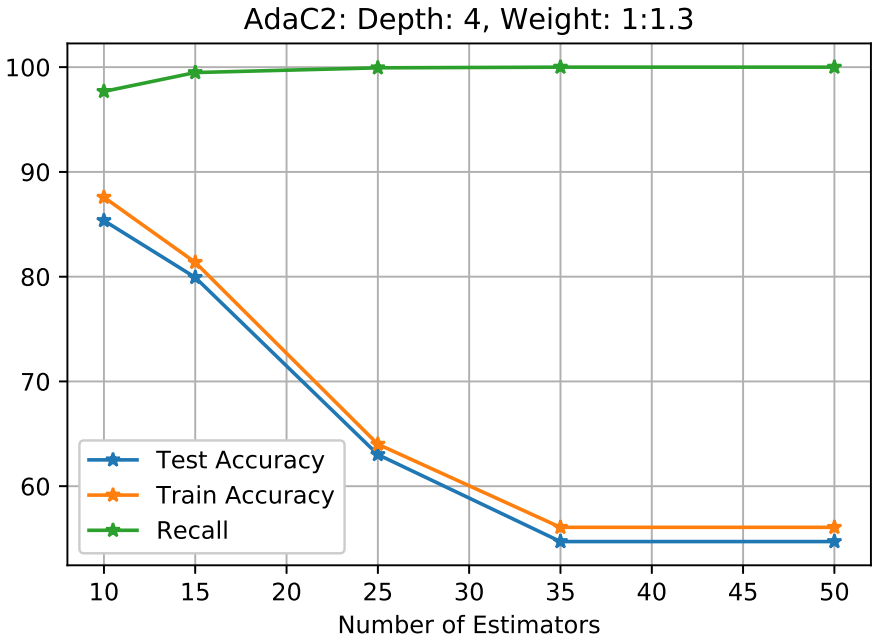

Figure 6.11: AdaC2 - Maximum depth of tree: 4

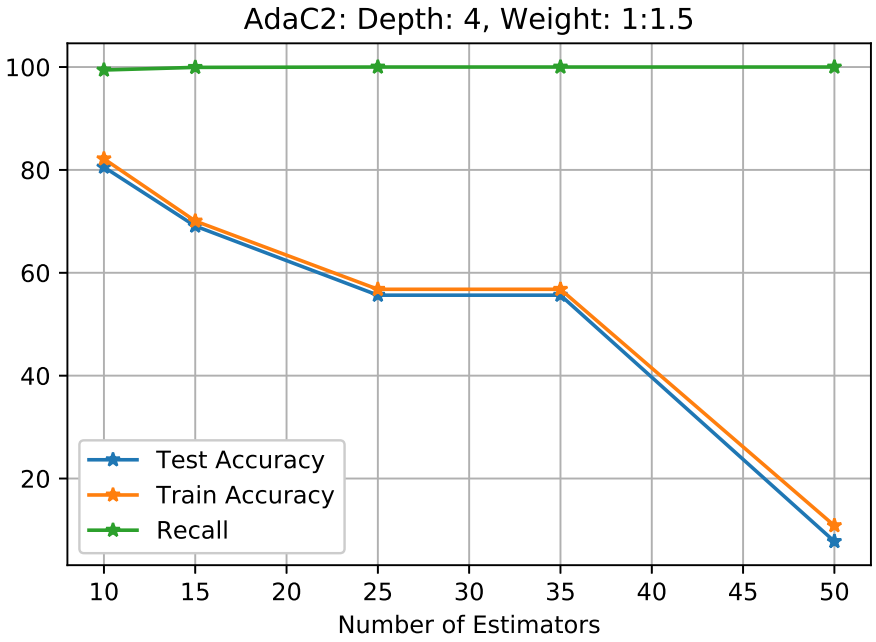

Figure 6.12: AdaC2 - Maximum depth of tree: 4

We similarly train the final cost-sensitive algorithm, AdaCost, with varying values of cost, number of weak learners and maximum depth of the decision trees. AdaCost algorithm requires $C_0, C_1 \in (0, 1]$, thus the classifiers are trained with $[C_0 : C_1]$ set as either $[0.2 : 0.5], [0.3 : 0.5]$ or $[0.4 : 0.5]$.

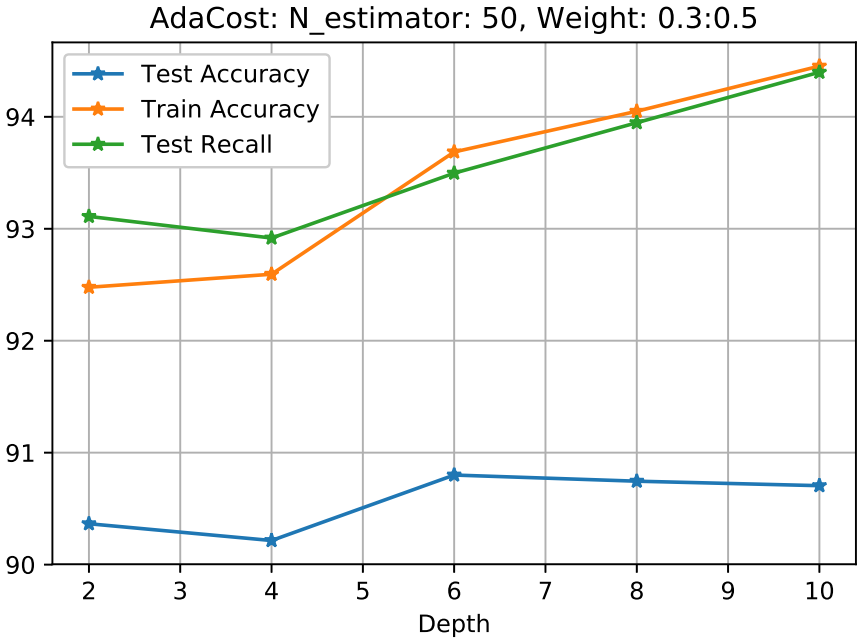

Figure 6.13: AdaCost - Number of weak learners: 50

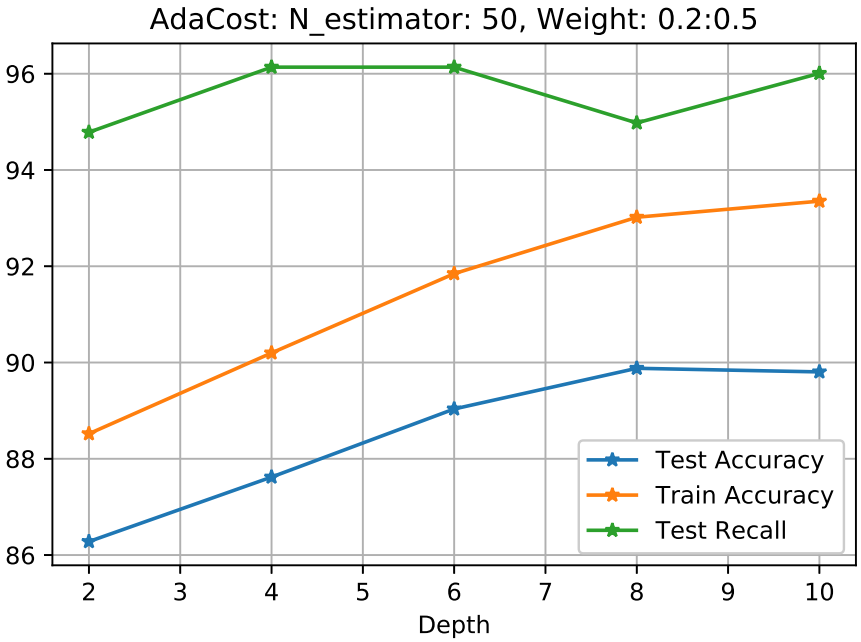

Figure 6.14: AdaCost - Number of weak learners: 50

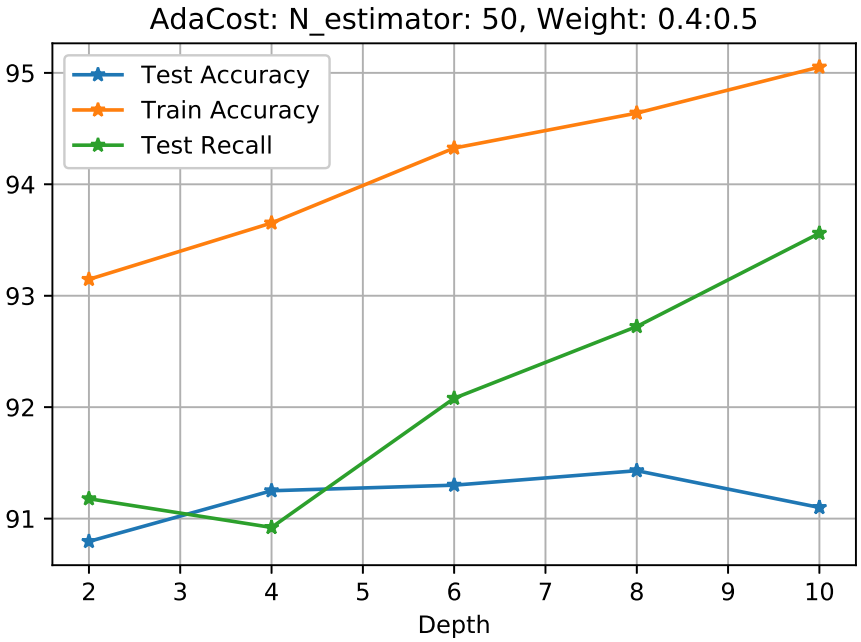

Figure 6.15: AdaCost - Number of weak learners: 50

The quantitative characteristics observed for AdaC2 and AdaCost are similar to the observations made for AdaC1. The recall improves with an increase in the cost for the minority class and with an increase in the complexity (depth) of the decision trees.

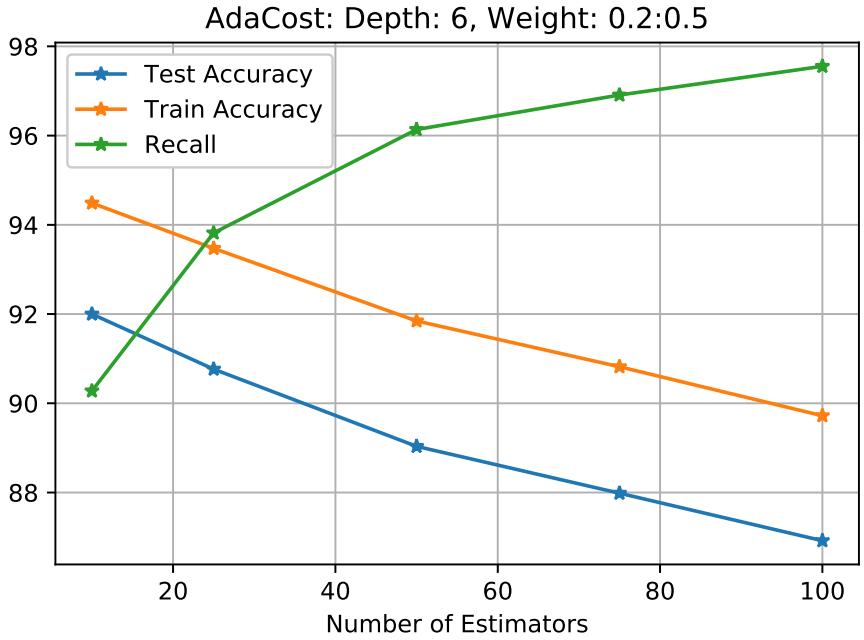

Figure 6.16: AdaCost - Maximum depth of tree: 6

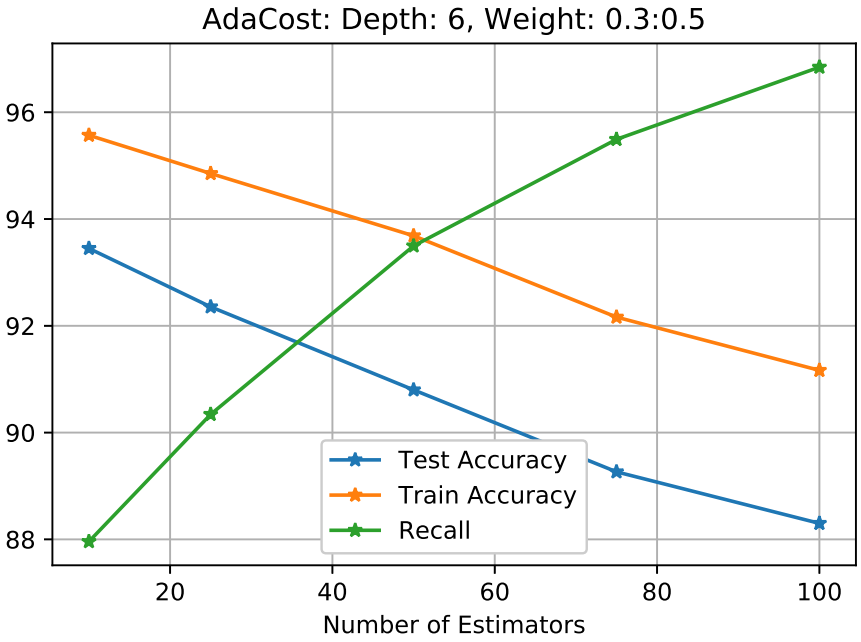

Figure 6.17: AdaCost - Maximum depth of tree: 6

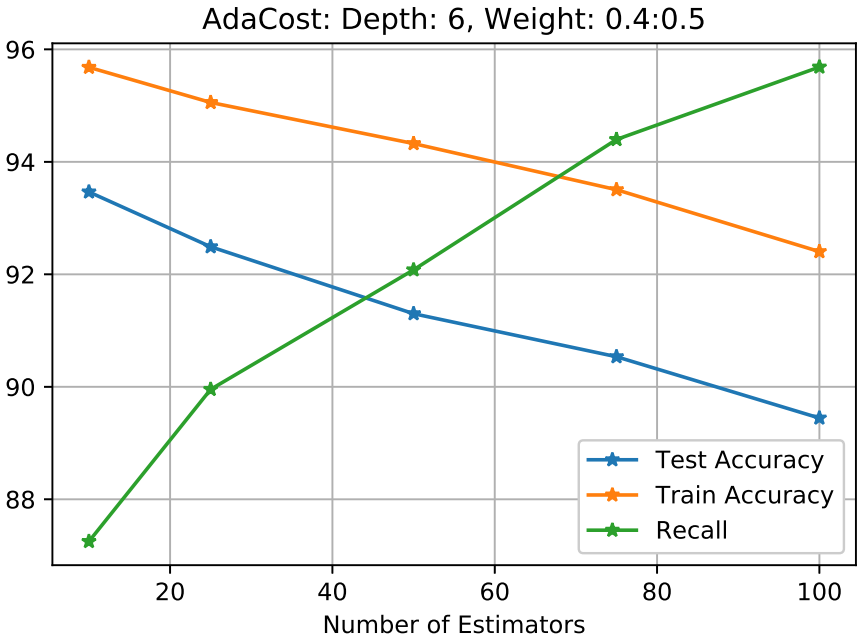

Figure 6.18: AdaCost - Maximum depth of tree: 6

We compare the algorithmic approaches to learning the imbalanced dataset with the data processing approaches. Since OverBagging increases the number of data samples resulting in an increased complexity during the training process, we implement the UnderBagging algorithm with ensemble learners for comparing the two approaches. UnderBagging is the process of subsampling the majority class samples and obtaining a balanced data set for training with no change on the testing data set. When comparing the results obtained for UnderBagging with the algorithmic approaches, we observe comparable results for recall but lower test accuracy values for UnderBagging since some majority class information is lost while subsampling. Figure 6.19 plots the training accuracy, testing accuracy and recall for UnderBagging classifier using ensemble of decision trees.

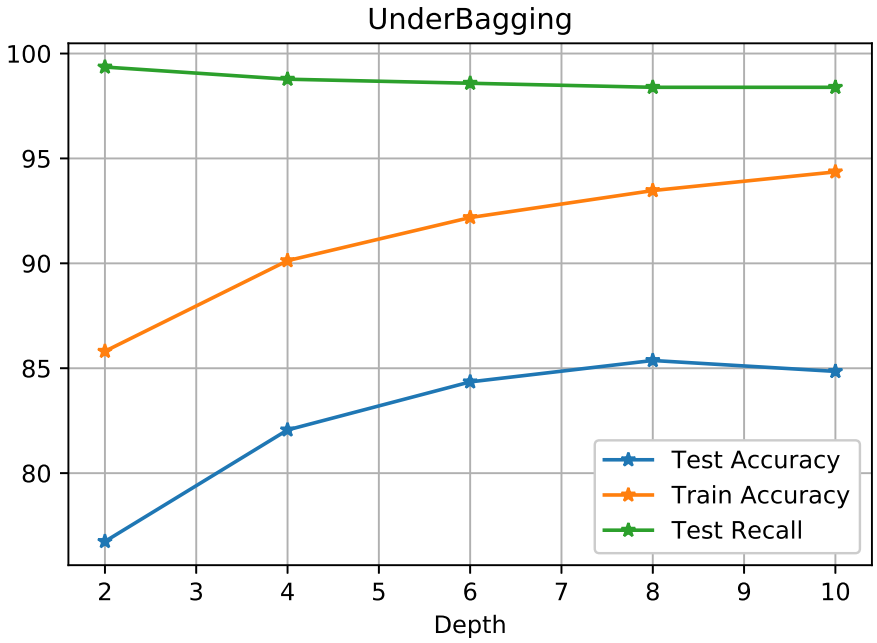

Figure 6.19: Underbagging

We implement SVM ensembles with 9 SVMs trained on a balanced data set with each individual SVM independently trained with the same samples of the minority class coupled with a different set of the majority class samples. We use two methods to obtain the majority set for

Table 6.1: SVM performance

	Random sampling		Sequential divided	
	Accuracy	Recall	Accuracy	Recall
1	92.06	94.7	76.61	97.48
2	92.08	94.84	77.7	97.68
3	92.2	94.39	91.91	94.84
4	92.03	94.78	92.47	91.63
5	92.05	94.6	91.22	93.68
6	92.0	94.9	85.83	98.58
7	92.15	94.78	42.2	100
8	92.11	94.39	39.27	100
9	92.11	94.59	36.03	100
Aggregate SVM	92.21	94.59	91.99	94.84

each SVM. Either by randomly sampling the good frames with replacement or by sequentially dividing the data set. The individual test accuracy, individual test recall, overall test accuracy and overall recall are presented in Table 6.1.

6.2 Labeling - Any

In this section we present the results for ensemble learners on the labeling technique *Any*. The frames are labeled as bad/good frame using the *Any* criterion for error window $[9K : 10K]$ PE cycle and the input variables are the error counts from the PE cycle $[7K : 8K]$. After labelling the 218435 frames, we obtain 183490 good frames and 34945 bad frames which are divided into 198435 training samples consisting of 166225 samples from good frames, 32210 samples from bad frames and 20000 testing samples comprising 17265 good frame and 2735 bad frame samples.

The results for AdaC1, AdaC2 and AdaCost with varying parameters as performed in section 6.1 are presented for *Any* labeled dataset. Figure 6.20, 6.21, 6.22, 6.23, 6.24 and 6.25 plots the performance curves for AdaC1 algorithm.

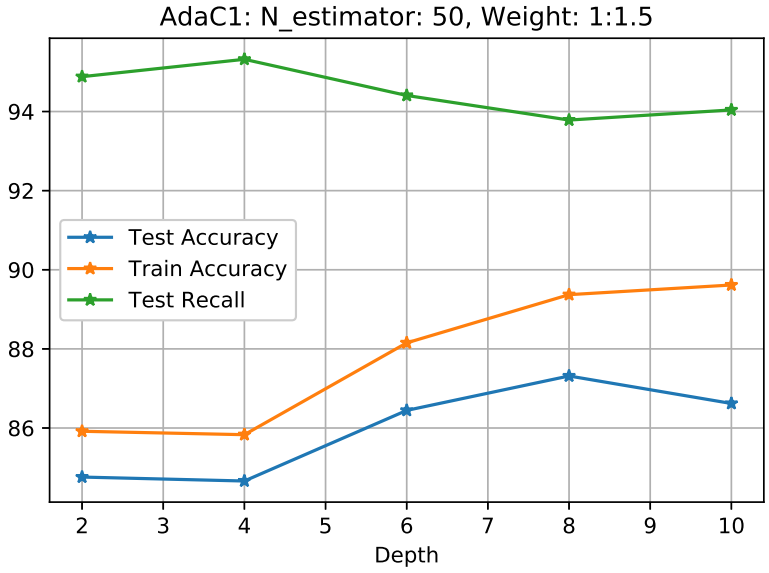

Figure 6.20: AdaC1 - Number of weak learners: 50

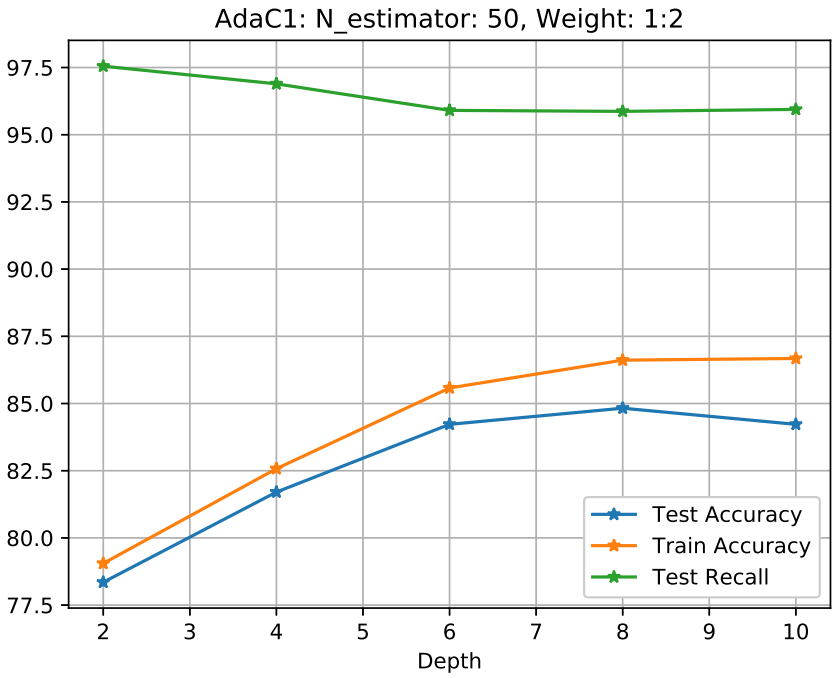

Figure 6.21: AdaC1 - Number of weak learners: 50

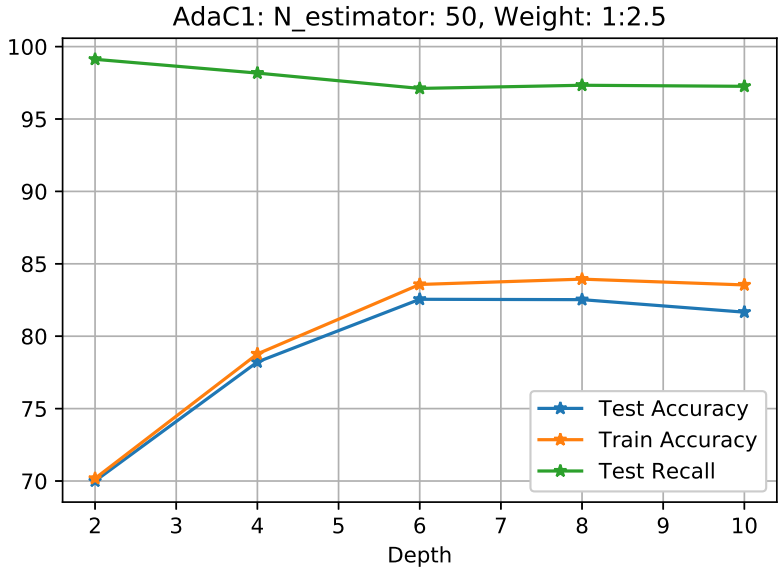

Figure 6.22: AdaC1 - Number of weak learners: 50

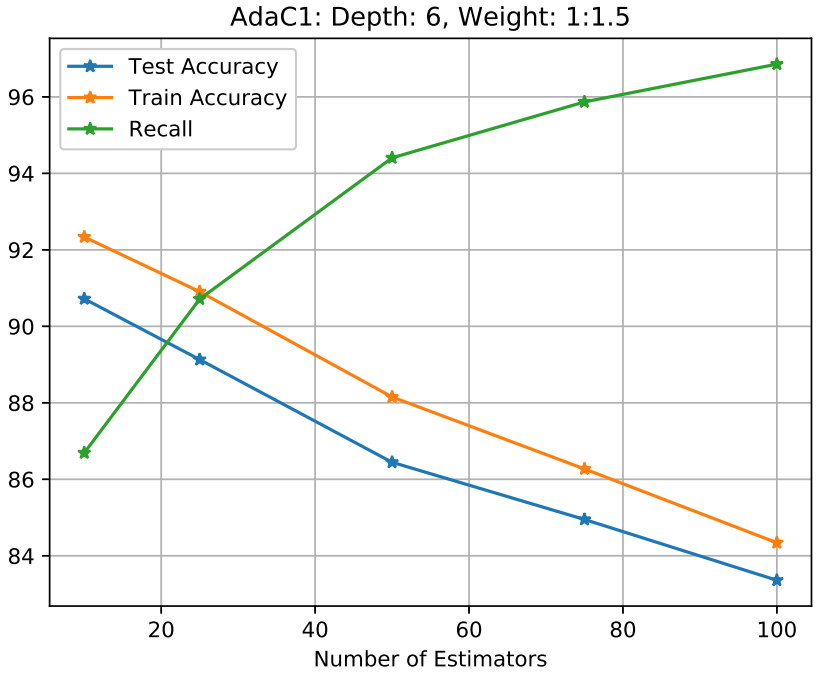

Figure 6.23: AdaC1 - Maximum depth of tree: 6

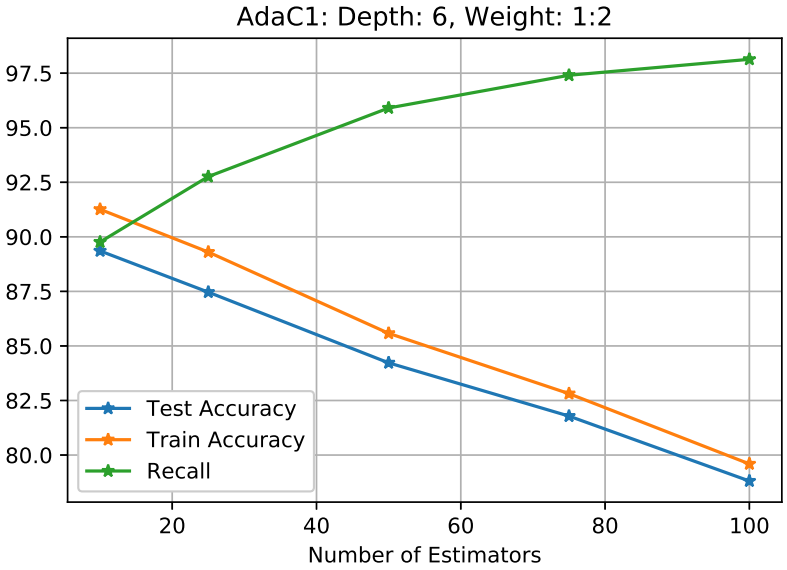

Figure 6.24: AdaC1 - Maximum depth of tree: 6

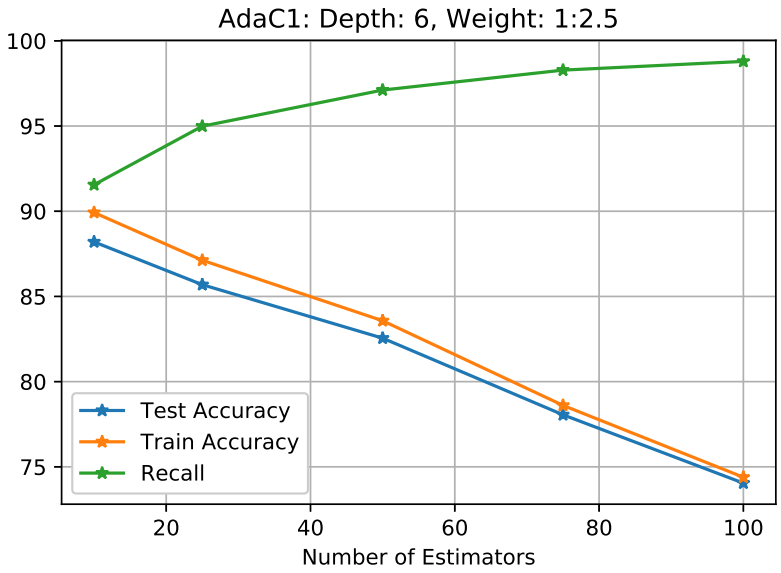

Figure 6.25: AdaC1 - Maximum depth of tree: 6

For AdaC2 algorithm on *Any* labeling, the parameters are set as: cost for bad class 1.1, depth of tree $2, 4, 6, 8, 10$ and number of weak learners $10, 15, 25, 35, 50$.

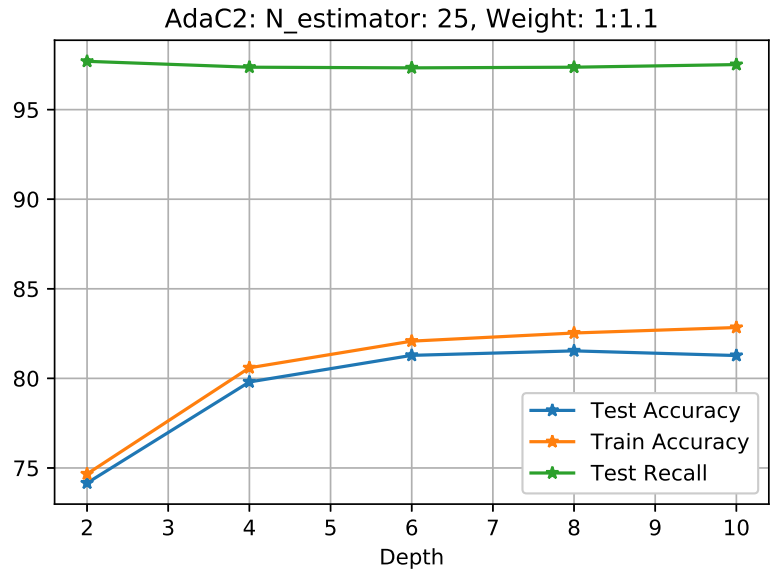

Figure 6.26: AdaC2 - Number of weak learners: 25

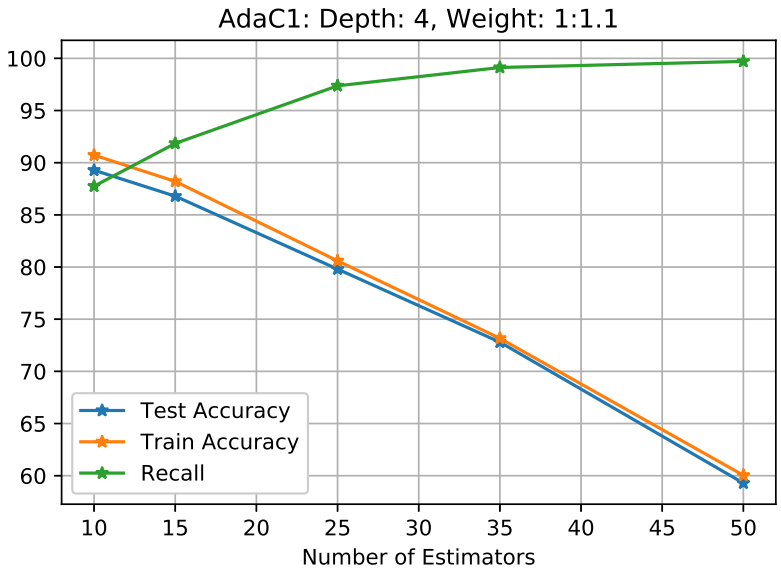

Figure 6.27: AdaC2 - Maximum depth of tree: 4

For AdaCost algorithm on *Any* labeling, the parameters are set as: cost ratio $0.3 : 0.5$, depth of tree $2, 4, 6, 8, 10$ and number of weak learners $10, 25, 50, 75, 100$.

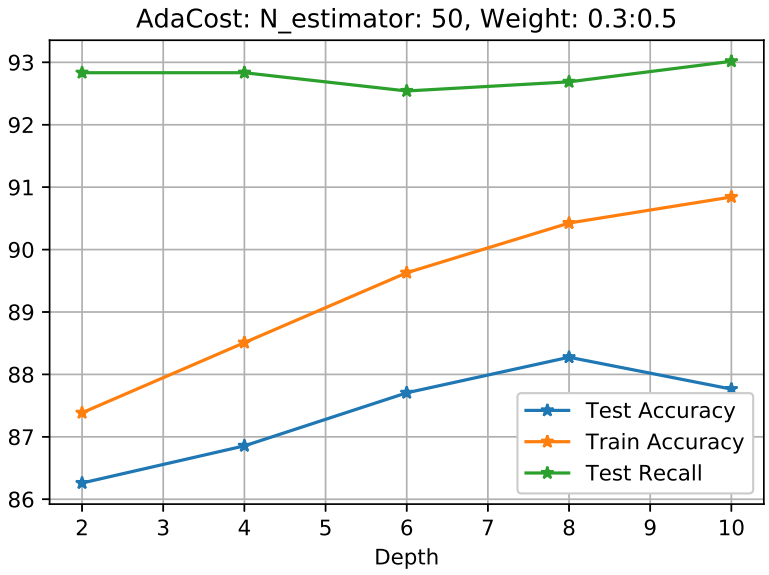

Figure 6.28: AdaCost - Number of weak learners: 50

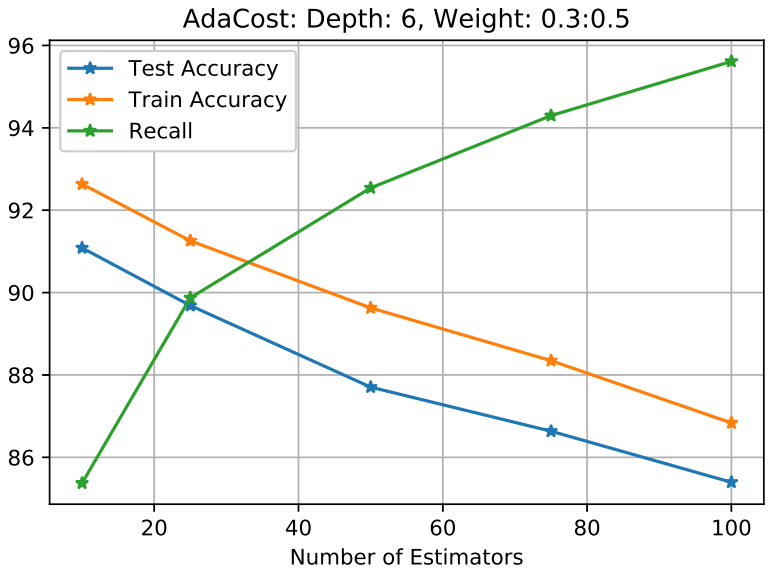

Figure 6.29: AdaCost - Maximum depth of tree: 6

6.3 CNN

This section presents the performance results for the neural network architecture. First we present the results of the network with and without including a fixed cost during training process. An observation was made while training the network with fixed cost. The network trained with equal costs assigned to both the classes had consistent results for test accuracy but the results for test recall values had significant variation. The accuracy and recall results for the neural network on different trials of training the network with random seed for initializing the network weights and performing a validation after every 10 epochs are shown in the table below.

Table 6.2: Test accuracy and test recall

Cost[1:1]	Trial-1	Trial-2	Trial-3	Trial-4
Accuracy	95.89	95.40	96.2	95.76
Recall	82.87	85.06	80.36	83.32

A similar variation in recall and accuracy was observed while including the validation and excluding the validation during the training process with a fixed seed to initialize the network weights. The Figure 6.30 below plots the test accuracy on training the network with and without validation for varying values of cost factor and Figure 6.31 plots the test recall for the same. It can be observed that the accuracy has a variation of up to 1%, whereas the recall has a variation of 10% initially and the variation in recall reduces as we increase the cost for minority class.

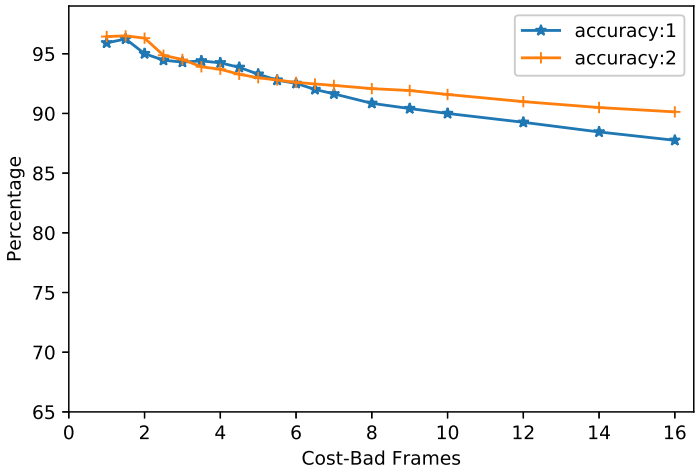

Figure 6.30: Accuracy results

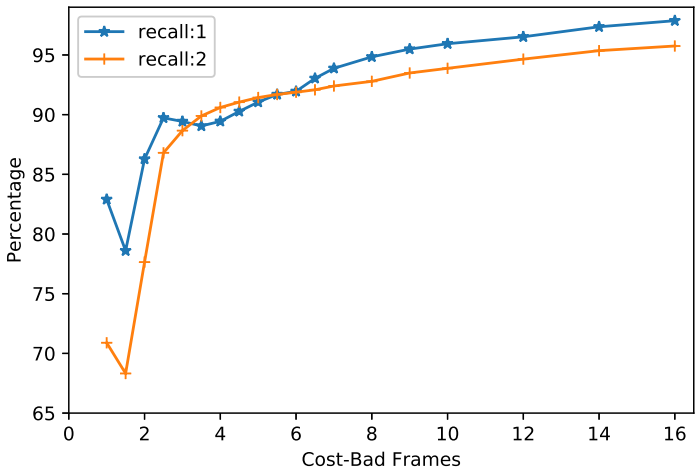

Figure 6.31: Recall results

We run multiple training trials for the network defined with varying cost for minority class to observe the consistency obtained in test results. The table below presents the accuracy and recall for the various trials. We can observe that as the cost on minority class increases, we obtain consistent results for both accuracy and recall compared to the results for lower cost. We trained the network with the cost for minority class set as the overall imbalance ratio, 8.13. Table

Table 6.3: Test accuracy and test recall with varying cost

Cost		Trial-1	Trial-2	Trial-3	Trial-4
[1:1]	Accuracy	95.89	95.40	96.2	95.76
	Recall	82.87	85.06	80.36	83.32
[1:2]	Accuracy	94.5	93.75	94.78	95.5
	Recall	88.41	90.34	87.25	84.80
[1:4]	Accuracy	91.54	92.83	92.41	92.31
	Recall	93.88	91.69	92.27	92.40
[1:6]	Accuracy	91.95	92.93	93.64	91.38
	Recall	93.24	91.50	90.53	94.33
[1:8]	Accuracy	91.89	91.85	91.38	91.63
	Recall	93.49	93.49	94.20	93.82
[1:10]	Accuracy	87.57	89.25	89.49	89.25
	Recall	98.13	96.55	96.26	96.52

6.4 presents the result for training the network with the cost assigned as the imbalance ratio and Figure 6.32 plots the training and validation loss for the network.

Table 6.4: Accuracy and recall: imbalance ratio

[1:8.13]	Trial-1	Trial-2	Trial-3	Trial-4
Accuracy	90.94	90.6	90.97	90.02
Recall	94.78	95.23	94.97	95.81

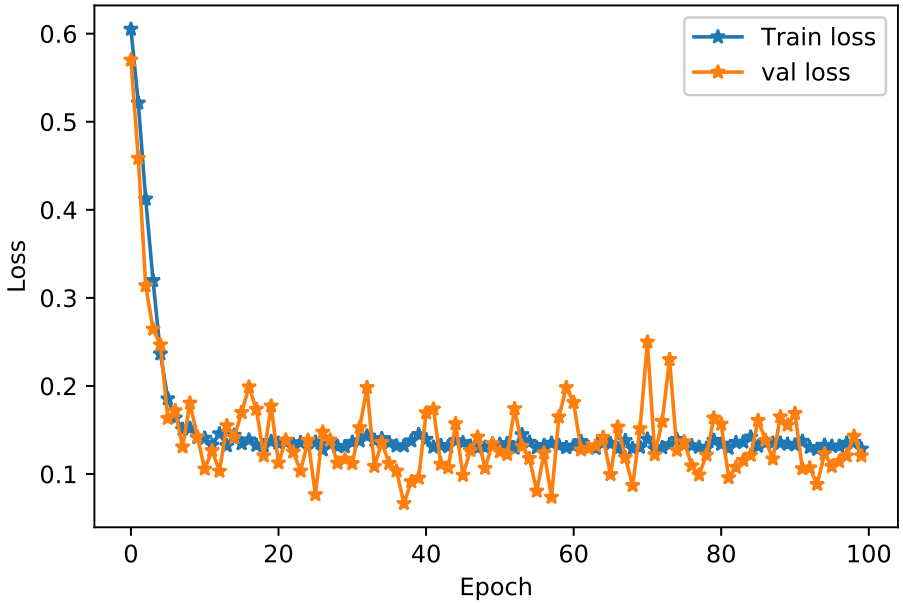

Figure 6.32: Network loss

Underbagging with neural network results in a recall of 96% but the test accuracy is 89%. Undersampling the majority class affects the overall accuracy, since some important samples belonging to the majority class might be skipped.

We present the results of training the network with adaptive cost for minority class. The cost is updated using the performance of the network over the mini-batch. Table 6.5 presents the test accuracy and test recall results for different trials.

Table 6.5: Accuracy and recall: adaptive cost

	Trial-1	Trial-2	Trial-3	Trial-4
Accuracy	92.7	91.88	92.9	93.12
Recall	91.69	93.66	91.5	91.24

6.4 Time Series Dataset

We present the results for training the LSTM using the time series dataset obtained by labeling using the third method mentioned in chapter 2. Understanding the training complexity for LSTM with large dataset, we train the model using time series data labeled as good page and bad page. Each page is labeled as a bad/good page for the PE cycle windows : $[7K : 8K]$, $[8K : 9K]$ and $[9K : 10K]$. The inputs to LSTM are bit error counts for windows $[5K : 6K]$, $[6K : 7K]$ and $[7K : 8K]$ for each page. The input to the LSTM is three dimensional array with the first dimension representing the batch size, second dimension represent the three PE cycle windows (time steps) and third dimension represents the bit error counts (variables) in the particular PE cycle window. The dataset is fed in LSTM in order for the network to use the information learnt from previous PE cycle windows to make the prediction on the current PE cycle.

We also train a cost sensitive classifier (AdaC1) with the dataset and compare the results with LSTM. For AdaC1 algorithm, along with the bit error counts the PE cycle information is included as an input variable. Table 6.6 presents the results for LSTM and AdaC1. We observe a

Table 6.6: Results time series data

	Accuracy	Recall
AdaC1[1:2.5]	96.6	99.1
LSTM	98.8	87

high accuracy from LSTM but the recall is lower compared to cost sensitive boosting algorithm.

Chapter 7

Conclusion

In this thesis, we have studied the dataset obtained from the 1Xnm TLC memory and performed various labeling methods to understand different failure modes. Further, using the labeled data we have implemented numerous learning techniques to predict the failures. We initially looked at classical machine learning techniques to predict the failures. The results lead us to the conclusion that the algorithmic techniques such as AdaC1, AdaC2 and AdaCost achieves an overall better performance compared to the data processing technique such as UnderBagging. We also implemented SVM ensembles for learning the imbalanced dataset which produce classifiers with good recall and accuracy. Varying the parameters and the cost, these classical machine learning techniques achieve different trade-off values for recall and accuracy which is a valuable information for the system architecture of the device. A better recall guarantees no loss of data since the bad pages are detected prior to the failure. However, increase in recall results in a decrease in correctly predicted good pages, and the mis-prediction of good pages results in loss of useful storage space.

We proceeded to explore various neural network based techniques with and without a cost. We observe that the network performance for a non cost sensitive model is consistent for the accuracy. We discover higher variation in the performance with respect to minority class and

lower recall results. The high variation in recall is not observed for a cost sensitive model when the costs are assigned with the knowledge of the overall imbalance in the dataset. As the cost factor increases, we observe the test recall improves with a slight reduction in the test accuracy. Finally, we implemented and compared the results for time series dataset with a classical cost sensitive algorithm and LSTM. We observe good accuracy results for LSTM while the cost sensitive algorithm exhibits higher recall values.

In this project, we have implemented and compared various learning techniques and their ability in learning an imbalanced dataset. The results have proved that learning imbalanced dataset requires modification in the machine learning techniques to include the cost. In future, we would like to explore strategies to optimize the cost per class while training the neural network. Currently the models make the predictions with only bit error counts as input variables, going forward we want to introduce other features like page type, location on the page in the block, read thresholds etc. We want to study and investigate algorithms to study retention failures caused due to charge leakage in flash memory.

Bibliography

[1] Guillaume Chevalier. The lstm cell.

[2] Robert E Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[3] Yanmin Sun, Mohamed S Kamel, Andrew KC Wong, and Yang Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, 40(12):3358–3378, 2007.

[4] Wei Fan, Salvatore J Stolfo, Junxin Zhang, and Philip K Chan. Adacost: misclassification cost-sensitive boosting. In *ICML*, volume 99, pages 97–105, 1999.

[5] Robert E Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[6] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[7] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484, 2011.

[8] Edward Y Chang, Beitao Li, Gang Wu, and Kingshy Goh. Statistical learning for effective visual information retrieval. In *ICIP (3)*, pages 609–612, 2003.

[9] Shohei Hido, Hisashi Kashima, and Yutaka Takahashi. Roughly balanced bagging for imbalanced data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2(5-6):412–426, 2009.

[10] Rong Yan, Yan Liu, Rong Jin, and Alex Hauptmann. On predicting rare classes with SVM ensembles in scene classification. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings (ICASSP'03)*, volume 3, pages III–21. IEEE, 2003.

[11] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer science & business media, 2013.

[12] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[13] Belur V Dasarathy. Nearest neighbor NN norms: NN pattern classification techniques. *IEEE Computer Society Tutorial*, 1991.

[14] Salman H Khan, Munawar Hayat, Mohammed Bennamoun, Ferdous A Sohel, and Roberto Togneri. Cost-sensitive learning of deep feature representations from imbalanced data. *IEEE Transactions on Neural Networks and Learning Systems*, 29(8):3573–3587, 2017.

[15] Yue Geng and Xinyu Luo. Cost-sensitive convolution based neural networks for imbalanced time-series classification. *arXiv preprint arXiv:1801.04396*, 2018.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.